Henry Joutsijoki ja Erkki Mäkinen (toim.)

Tietojenkäsittelytieteellisiä tutkielmia
Talvi 2018

TAMPEREEN YLIOPISTO

INFORMAATIOTIETEIDEN YKSIKÖN RAPORTTEJA 59/2018

TAMPERE 2018

TAMPEREEN YLIOPISTO
INFORMAATIOTIETEIDEN YKSIKÖN RAPORTTEJA 59/2018
TAMMIKUU 2018

Henry Joutsijoki ja Erkki Mäkinen (toim.)

Tietojenkäsittelytieteellisiä tutkielmia
Talvi 2018

ISBN 978-952-03-0654-0 (pdf)

ISSN-L 1799-8158
ISSN 1799-8158

Sisällysluettelo

Tietojärjestelmät osana organisaation kokonaisarkkitehtuuria

Joni Malmberg

Tiivistelmä.
Kokonaisarkkitehtuuri on melko laaja ja abstrakti termi. Tässä tutkielmassa käsitellään kokonaisarkkitehtuurin kokonaisuutta tietojärjestelmän näkökulmasta katsottuna. Tutkielmassa tarkastelllaan kokonaisarkkitehtuurin kehittämisestä koituvia hyötyjä ja mahdollisia ongelmakohtia. Osa näistä ongelmista voidaan välttää valitsemalla kyseiselle organisaatiolle ja sen tarpeille sopiva kokonaisarkkitehtuurikehys. Oikean kehyksen valitseminen organisaation kokonaisarkkitehtuurin luomiseen on yksi suunnitteluvaiheen tärkeimmistä osista, sillä valmiin kokonaisuuden vaihtaminen toiseen on hyvin hankalaa ja aikaa vievää. Suomessa on julkista hallintoa varten kehitetty oma JHS 179 -suositus kokonaisarkkitehtuurikehyksestä.

Kokonaisarkkitehtuurin hyötyjä on tutkittu varsin paljon. Osaa näistä tutkimuksista ei voida kuitenkaan pitää luotettavana, sillä kaikissa näissä ei ole ollut tarpeeksi laajaa aineistoa käytettävissä. Kokonaisarkkitehtuurin on kuitenkin tarkoitus muun muassa mahdollistaa organisaatiolle paremman ja kustannustehokkaamman tavan uusien tietojärjestelmien käyttöönotolle.

Tietojärjestelmät eivät joka organisaatiossa ole keskeisessä roolissa, mutta esimerkiksi sairaalaympäristössä on käytössä monia kriittisiä sovelluksia tai tietojärjestelmiä. Tämän kaltaisissa organisaatioissa yksi kokonaisarkkitehtuurin haasteista onkin saada nämä tietojärjestelmät toimimaan saumattomasti keskenään.

Avainsanat ja -sanonnat: Kokonaisarkkitehtuuri, tietojärjestelmät, organisaatiot.

1. Johdanto

Tietojärjestelmät ovat yhä tärkeämpi osa organisaatioiden toimintaa jatkuvasti digitalisoituvassa maailmassa. Tästä huolimatta kaikilla organisaatioilla ei ole selkeää kuvaa omista tietojärjestelmistään. Koska tietojärjestelmät ovat tärkeitä organisaatioille, tämänkaltainen oman organisaation toimintojen ja prosessien selkeän kokonaiskuvan puuttuminen voi aiheuttaa turhaa työtä tai ylimääräisiä kuluja. Lisäksi näiden ongelmien takia on luonnostaan noussut tarve luoda selkeällä tavalla kokonaisuus oman organisaation toiminnasta. Tämä taas tarkoittaa nimenomaan kokonaisarkkitehtuurin nousemista suureen rooliin organisaatioissa.

Tutkielman yhtenä tarkoituksena on selvittää mahdollisia ongelmakohtia kokonaisarkkitehtuurissa sekä tietojärjestelmissä. Tutkielma myös pyrkii tuomaan paremmin esille mahdollisia hyötyjä kokonaisarkkitehtuurin kehittämisestä ja syitä siihen, miksi sitä tarvitaan. Tarkoituksena on myös pureutua tietojärjestelmien rooliin ja sen merkitykseen kokonaisarkkitehtuurin toiminnan kannalta.

Tutkielmassani käytän esimerkkinä kokonaisarkkitehtuurin käytöstä terveydenhuollon järjestelmiä ja sairaalaympäristöä. Tämä valinta on kytköksissä työhöni monikansallisen suuryrityksen järjestelmäasiantuntijana. Olen työni puolesta paljon tekemisessä kyseisen sektorin IT-asioiden kanssa ja näin ollen niistä myös hyvin kiinnostunut. Perustelen tätä esimerkkivalintaani myös sillä, että kokonaisarkkitehtuurin merkitys nousee erityisen tärkeäksi juuri sairaalaympäristöissä, joissa on käytössä lukuisia jopa ihmishengen kannalta kriittisiä sovelluksia.

2. Kokonaisarkkitehtuuri

Zachmanin [1987] artikkelia ”A Framework for Information Systems Architecture” sekä hänen kehittämäänsä Zachman-kehystä pidetään yleisesti nykyisen kokonaisarkkitehtuurin perustana. Kyseisen kokonaisarkkitehtuurikehyksen jälkeen onkin luotu lukemattomia uusia kehyksiä muun muuassa eri aloille ja myös monille eri valtioille löytyy omansa. Esimerkiksi Suomen julkinen hallinto on luonut oman suosituskehyksensä omien tarpeidensa mukaisesti. Tätä suositusta kutsutaan JHS 179 -suositukseksi[1].

Kokonaisarkkitehtuurille on hankala antaa täsmällistä määritelmää, sillä kyseessä on varsin abstrakti termi. Lisäksi kokonaisarkkitehtuurin on määritelty tarkoittavan kahta eri asiaa. Esimerkiksi Kaisler ja muut [2005, 1] toteavat, että kokonaisarkkitehtuurilla voidaan tarkoittaa paitsi varsinaista toteutusta, myös siihen liittyviä dokumentointeja. Tutkielmassani kuitenkin lähtökohtaisesti viittaan kyseisellä termillä itse toteutukseen. Eri lähteissä kyseistä termiä määritelläänkin hieman eri tavoin, mutta niissä kuitenkin toistuu usein se, että kokonaisarkkitehtuuri tavallaan yhdistää koko organisaation toiminnan.

Schekkerman [2004, 13] on kuvannut kokonaisarkkitehtuurin eräänlaisena yrityksen *kokonaisvaltaisena määritelmänä*. Tämän tulkinnan mukaan kokonaisarkkitehtuuri nähdään tietynlaisena pääsuunnitelmana, jonka tarkoituksena on saada yrityksen kaikki eri osa-alueet, kuten esimerkiksi tietojärjestelmät ja liiketoimintaosastot toimimaan yhteistyössä keskenään. [Emt.]

[1] JHS tulee sanoista *julkisen hallinnon suositus*.

Toisaalta kokonaisarkkitehtuurin on myös esitetty olevan enemmänkin ne työkalut, prosessit ja oikeanlaiset rakenteet, joilla saavutetaan organisaation laajuinen yhtenäinen IT-arkkitehtuuri. Tämän IT-arkkitehtuurin tavoitteena on saada integroitua organisaation IT toimimaan yhteistyössä liiketoimintapuolen kanssa. [Kaisler *et al*. 2005, 1.]

Näiden kahden toisistaan poikkeavan määritelmän kautta piirtyy esille kokonaisarkkitehtuuri-käsitteen abstrakti luonne. Ensimmäinen edellä mainituista määritelmistä korostaa kokonaisarkkitehtuuria *koko* organisaation toiminnan näkökulmasta, kun jälkimmäinen liittää sen lähinnä IT-palveluihin ja liikeelämän toimintoihin. Nämä kaksi erilaista määritelmää tuovat hyvin esille eroja siinä, kuinka laajasti tai suppeasti käsitettä *kokonaisarkkitehtuuri* voidaan tulkita.

Tulkitsen termin määrittämisen vaikeuden tarkoittavan sitä, että kokonaisarkkitehtuurissa on kyse enemmänkin siitä, mitä sillä halutaan organisaatiossa saavuttaa. Kokonaisarkkitehtuurin tarkoituksena on selkeyttää organisaation toimintaa. Tähän ei sisälly pelkästään organisaation IT-laitteet tai -järjestelmät, eivätkä ne oikeastaan ole edes kokonaisarkkitehtuurin ydin, vaan kokonaisarkkitehtuurin avulla halutaan saavuttaa parempi ja selkeämpi ymmärrys organisaation omista toiminnoista, prosesseista ja liittymistä. Tämän ratkaisemiseen onkin tarjolla monia erilaisia kokonaisarkkitehtuurikehyksiä.

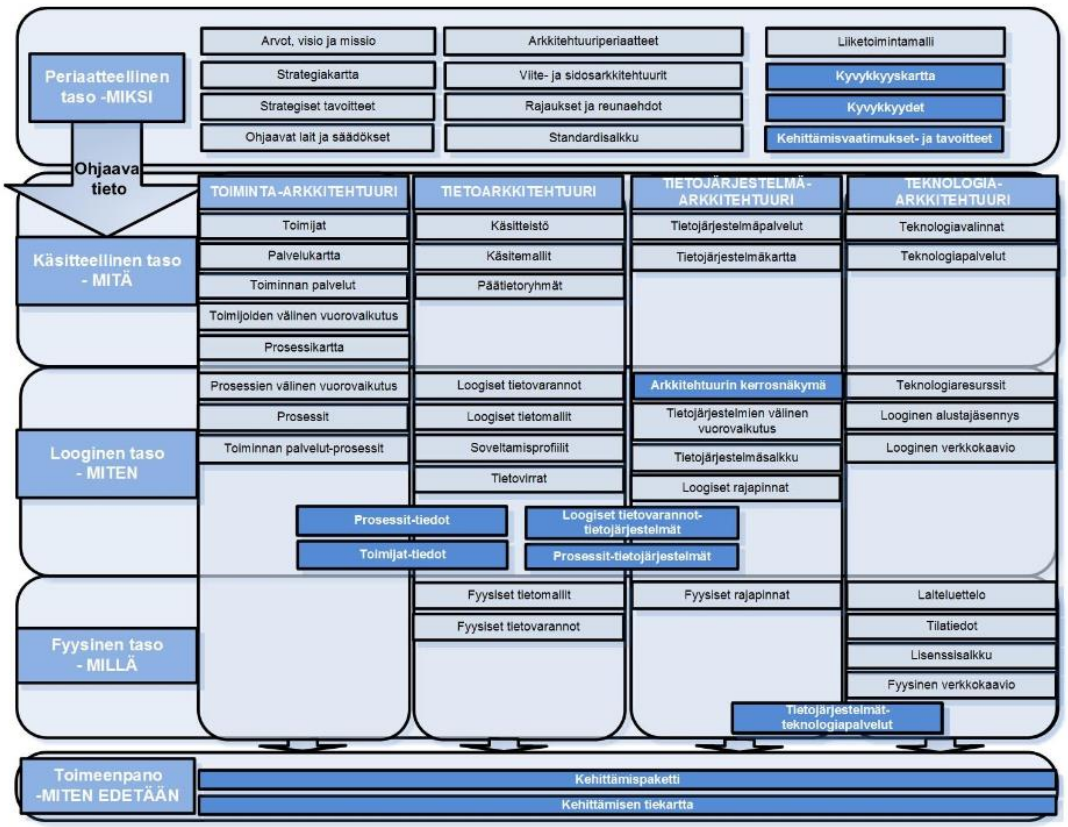
2.1. Erilaisia kokonaisarkkitehtuurikehyksiä

Erilaiset kehykset antavat valmiuden luoda kokonaisarkkitehtuurikokonaisuuksia tarjoamalla käyttöön sopivia menetelmiä ja työkaluja arkkitehtuurin osien rakentamiseen. Lisäksi kehykset ottavat kantaa siihen, miten nämä osat saadaan yhdistettyä yhdeksi kokonaisuudeksi. Käytännössä nämä kehykset siis sisältävät listan suositelluista standardeista ja helposti mukautuvista sovelluksista tietojärjestelmiä varten. [Shah and El Kourdi 2007, 37.] Yhdenlaiset kehykset sopivat eri alojen organisaatioille paremmin kuin toiset. Esimerkiksi terveydenhoitopuolella on tehty useita tutkimuksia eri kehysten sopivuudesta alan organisaatioihin [Purnawan and Surendro 2016]. Tämänkaltaisissa tutkimuksissa tutkijat pisteyttävät erilaisia kehyksiä organisaation tarpeiden mukaisesti. Nämä tutkimukset eivät kuitenkaan rajoitu pelkästään terveydenhoitoon, vaan niitä on tehty useammalta eri alalta.

Näkökulmiin perustuvissa kehyksissä (perspective frameworks) jaetaan koko organisaatio yksinkertaisiin näkökulmiin, jotka tarjoavat perusteet arkkitehtuurin varsinaiseen mallintamiseen. Nämä näkökulmat asetellaan viitekehykseen ruudukkomatriisin mukaisesti, joissa yhdelle akselille lisätään kysymyssanoja, kuten *mitä*, *miksi*, *koska* ja *kuka*. Toiselle akselille puolestaan lisätään roolit, esimerkiksi *suunnittelija*. Kyseiset ruudukot yhdistetään eri roolien näkö-

kulmilla kysymyssanoihin. Vaikkapa arkkitehdin roolille voidaan siis lisätä mitä-kysymykseen esimerkiksi tietojärjestelmän tekninen kuvaus, ja kuka-kysymykseen tietojärjestelmän eri osien vastaavat henkilöt. Tämänkaltainen arkkitehtuurikuvauksen viitekehys on myös yksi esimerkki kokonaisarkkitehtuurikehysten tarjoamista työkaluista. [Gerben *et al*. 2013, 2.]

Kuvassa 1 on kuvattuna aiemmin mainittu suomalaisen julkishallinnon kokonaisarkkitehtuurin viitekehys. Kyseinen viitekehys on Suomen julkisen hallinnon suositusten mukainen ja kuvasta voidaan nähdä, että JHS 179:n kuvaama viitekehys on hieman erilainen kuin mitä ylempänä annettu peruskuvaus antaisi ymmärtää. Tämä on osaltaan valittu täsmentämään sitä, että erilaisilla organisaatioilla on tarve omanlaiselle kokonaisarkkitehtuurille, jonka suunnitteluun puolestaan vaaditaan soveltuva kokonaisarkkitehtuurikehys.



Kuva 1. JHS 179:n mukainen arkkitehtuurikuvauksen viitekehys [JUHTA, 29].

Kuvassa 1 x-akselilla on nähtävillä neljä eri näkökulmaa (toiminta-, tieto-, tietojärjestelmä- ja teknologia-arkkitehtuurinäkökulmat), jotka julkinen hallinto on kokenut tärkeäksi. JHS 179 -suosituksissa [JUHTA 2017, 30] y-akselille kuuuvia kysymyssanoja (miksi, mitä, miten, millä ja miten edetään) kuvataan puolestaan seuraavasti:

- ”Periaatteellinen taso ohjaa suunnittelua ja kuvaamista (vastaa kysymykseen *miksi*).
- Käsitteellinen taso kuvaa tarpeita ja palveluja (vastaa kysymykseen *mitä*).
- Looginen taso kuvaa rakenteita (vastaa kysymykseen *miten*).
- Fyysinen taso kuvaa ratkaisuja (vastaa kysymykseen *millä*).
- Toimeenpanon taso kuvaa toimeenpanon keinoja (vastaa kysymykseen *miten edetään*).”

Viittauksiin perustuvissa kehyksissä (reference frameworks) kuvataan rakenteet, asiat ja suhteet näiden komponenttien välillä arkkitehtuurisessa kontekstissa. Tähän lisätään vielä suhteisiin liittyvät organisaation säännöt ja arvot, ja tätä kautta saadun kaavion avulla voidaan luoda sopiva kokonaisarkkitehtuuri. [Graves 2009, 33.]

Ontologiaan perustuvat kehykset (ontology frameworks) ovat huomattavan erilaisia kuin muut. Nämä eivät tarjoa niin selkeitä malleja etenemiseen kuin muut kokonaisarkkitehtuurikehykset. Ontologiaan perustuvissa kehyksissä panostetaan enemmän entiteettien ja perspektiivien välisiin suhteisiin. Tämänkaltainen kehys käytettynä organisaation kokonaisarkkitehtuurissa on muokkautuvampi ja monipuolisempi kuin muut kehykset. [Graves 2009, 34.]

2.2. Kokonaisarkkitehtuurin hyödyt

Vaikka kokonaisarkkitehtuuri tuo organisaatiolle monenlaisia hyötyjä, kaikki eivät kuitenkaan koe sitä välttämättä tarpeelliseksi. Esimerkiksi Lemmetti ja Pekkola [2012, 171] päätyvät siihen lopputulokseen, että joillekin organisaatioille kokonaisarkkitehtuuri on vain *pakollinen rutiini*. Toisille taas se voi olla työkalu, joka auttaa tekemään kustannustehokkaammin sellaiset työt, jotka ilman kokonaisarkkitehtuuria tulisivat tehdyksi resursseja kuormittaen. [Lemmetti ja Pekkola 2012, 171.] Gofore Oy:n palveluarkkitehti Juuso Tuomola [2017] on puolestaan todennut: ”Ainoa toiminnan suunnittelu ja johtamisen väline, jolla voidaan kattavasti, järjestelmällisesti ja pysyvästi varmistaa, että rakennettavat IT-ratkaisut todella vastaavat toiminnan tarpeita.” Tämä on melko vahva väite, mutta tässä luvussa käsiteltävät hyödyt antavat mielestäni ymmärtää, että kokonaisarkkitehtuurilla pystytään saavuttamaan kyseiset tavoitteet. Kuitenkaan kysymykseen siitä, onko tämä todella ainoa tapa näiden tavoitteiden saavuttamiseksi, ei tämän tutkielman puitteissa ole mahdollista antaa vastausta.

Niemi ja Pekkola [2013] tuovat esille laadun tarpeen myös kokonaisarkkitehtuurin kohdalla. He toteavat, että kokonaisarkkitehtuuri ei itsessään riitä, vaan se tulisi olla laadukkaasti toteutettu, jotta organisaation on mahdollista

saada siitä hyötyä. Tämä ajatus on hieman ristiriidassa sen suhteen, että kokonaisarkkitehtuurin laatuominaisuuksia ei ole oikeastaan tutkittu. [Niemi ja Pekkola 2013, 3878.] Näihin laatuominaisuuksiin liittyvien kysymysten lisäksi on esitetty, että edes täydellisesti rakennetusta kokonaisarkkitehtuurista ei ole hyötyä, mikäli kokonaisarkkitehtuurin suunnittelun aikana rakennettuja prosesseja tai palveluja ei käytetä [Niemi ja Pekkola 2017, 315]. Tässä korostuukin organisaation tietohallinnon merkittävä rooli. Tietohallinnon tehtävänä on pitää huolta siitä, että organisaation työntekijät ovat tietoisia uusista prosesseista. Tässä kohtaa on hyvä korostaa koko organisaation sisäistä kommunikaatioita. Tulkitsen näiden asioiden olevan yhteydessä aiemmin mainittuihin organisaaioiden erilaisiin käsityksiin koskien kokonaisarkkitehtuuria.

Niemi ja Pekkola [2009] ovat todenneet, että kokonaisarkkitehtuurien hyötyjen tutkiminen on hankalaa. Yhtenä syynä tähän he ehdottavat kokonaisarkkitehtuurin jatkuvaa muutosprosessia. Jatkuvat muutokset saattavat tehdä mahdollisten hyötyjen mittaamisesta hyvin hankalaa [Niemi ja Pekkola 2009, 1]. Hyötyjen tutkimista varten monet tutkijat ovatkin ehdottaneet tapoja mitata niitä etuja, joita organisaatiot kokonaisarkkitehtuurista saavat. Niemi ja Pekkola [2009] selvittävätkin tutkimusten käyttämiseksi Delonen ja McLeanen onnistumismallia (Delone and McLean Information System Success Model). He toteavat tämän olevan yksi erittäin käyttökelpoinen malli kokonaisarkkitehtuurin hyötyjen selvittämiseksi [Niemi ja Pekkola 2009, 7].

Varsinaisista hyödyistä yksi keskeisimmistä pohjautuu siihen, että kokonaisarkkitehtuuri mahdollistaa paremman ja kustannustehokkaamman tavan uusien tietojärjestelmien käyttöönotolle tai nykyisten kehittämiselle. Graves [2009] käsittelee hyötyä, joka liittyy yksinkertaisesti siihen, että kokonaisarkkitehtuurin avulla organisaation johto saa tarkemman kuvan organisaationsa toiminnoista ja prosesseista. Tämä taas helpottaa johtoa tekemään koko organisaatiota koskevia päätöksiä ja suunnitelmia.

Tamm ja muut [2011] ovat tehneet kirjallisuuskatsauksen kokonaisarkkitehtuurin hyödyistä. He listasivat myös viisi sellaista kokonaisarkkitehtuurin hyötyä organisaatioille, joihin oli kirjallisuudessa eniten viitattu. Näitä olivat: 1) reagointien lisääntyminen ja muutosten ohjaaminen, 2) päätöksenteon parantuminen, 3) viestinnän ja yhteistyön parantuminen, 4) pienemmät kulut ja 5) liiketoiminnan ja IT:n yhteensovittaminen (Business-IT Alignment) [Tamm *et al*. 2011, 146]. Kaikkia näitä kohtia ei kuitenkaan oltu artikkelissa tarkemmin selitetty. Itse kuitenkin tulkitsen, että kohdassa yksi *reagointien lisääntymisellä* tarkoitetaan koko organisaation muutoskyvyn nousemista. Tämä tarkoittaa ymmärrykseni mukaan sitä, että pystytään paremmin reagoimaan myös organisaation ulkopuolelta tuleviin paineisiin. Tällaisia organisaation ulkopuolelta

tulevia paineita voivat olla esimerkiksi uusien teknologioiden käyttöönottamisen tai vaikka toisen yrityksen konkurssin aiheuttaman markkinoiden tyhjiön nopeampaan hyväksikäyttämiseen. Kohtien kaksi ja kolme tulkitsen puolestaan tulevan suoraan kokonaisarkkitehtuurin pääasiallisesta tarkoituksesta eli koko organisaation toiminnan ja kokonaiskuvan paremmasta hahmottamiskyvystä. Kun tiedetään, mitä tällä hetkellä tehdään ja missä mennään, tämän voidaan katsoa myös helpottavan päätöksentekoa varsinkin organisaation sisäisissä asioissa. Parempi kokonaiskuva organisaatioista myös luonnollisesti helpottaa sisäistä viestintää huomattavasti, kun tiedetään, keneen ja mihin osastoon tulisi olla yhteydessä tietyistä asioista. Kohdan neljä pienemmät kulut ovat myös helposti selitettävissä jo pelkästään kokonaisarkkitehtuurihankkeen yhteydessä tapahtuvasta prosessien kehityksellä. Tehokkaammat prosessit takaavat myös kustannussäästöt. Kohdan viisi Tamm ja muut [2011, 152] totesivat tarkoittavan sitä, että organisaation kaikki osa-alueet ovat tietoisia myös liiketoimintansa tavoitteista ja suunnitelmista.

Kyseisessä artikkelissa tuodaan myös esille, että vain muutamissa tutkimuksissa oli tutkittu kokonaisarkkitehtuurin suunnitteluvaiheen hyötyjä [Tamm *et al.* 2011, 144]. Suunnitteluprosessissa olennainen osa on olemassa olevien järjestelmien, toimintojen ja kaiken muun merkityksellisen tiedon dokumentointi. Tältä osin on jo siis mahdollisesti luvassa tärkeitä tietoja organisaation johdolle ja työntekijöille parempien dokumenttien muodossa.

2.3. Kokonaisarkkitehtuurin ongelmia

Organisaatioiden saamista monista erilaisista hyödyistä huolimatta kokonaisarkkitehtuuri ei kuitenkaan ole täysin ongelmatonta. Tästä huolimatta haasteita tai haittoja tutkivia artikkeleita aiheesta löytyy hyvin vähän. Lisäksi osa näistä on jo 2000-luvun alkupuolelta. Ajankohtaisen tutkimustiedon puuttuessa on hyvin todennäköistä, että kaikkia aiemmin esitetyistä ongelmista ei mahdollisesti nähdä enää niin suurina haasteina. Esimerkiksi vielä kymmenen vuotta sitten katsottiin, ettei kaikkia kokonaisarkkitehtuurikehyksiä ole kyetty suunnittelemaan niin, että niiden avulla olisi mahdollista reagoida helposti liike-elämän strategisiin muutoksiin [Shah and El Kourdi 2007, 40]. Tämä ajatus on myös ristiriidassa nykyisen hyötyjä koskevan ymmärryksen kanssa, koska kuten aiemmin toin esille, yksi kokonaisarkkitehtuurin tarkoituksista on nimenomaan parantaa organisaation reagointia myös liike-elämän muutoksiin.

Tämän lisäksi näiden kehysten käyttäminen ja kokonaisarkkitehtuurin kehittäminen voi olla joissain tapauksissa hyvinkin hankalaa. Lisää ongelmia tulee myös sen myötä, että osa kehyksistä ei perustu olioihin. Tästä taas seuraa ongelmia suunnittelun suhteen, kun organisaatiot eivät voi käyttää tuttua

UML-mallinnusta kokonaisarkkitehtuurin suunnitteluun. [Shah and El Kourdi 2007, 40.]

Edellä mainitut ongelmat ovat kuitenkin enemmän kehyksiin kuin suoranaisesti kokonaisarkkitehtuuriin liittyviä kysymyksiä. Kokonaisarkkitehtuurin suunnittelun ongelmallisuus on kuitenkin iso kysymys, joka liittyy suoraan organisaatioihin. Mitä isompi organisaatio ja mitä huonommin informaatio on aikaisemmin dokumentoitu, sitä vaikeampi on toteuttaa kokonaisarkkitehtuurin onnistunutta käyttöönottoa.

Kokonaisuutena voidaan kuitenkin todeta, että on tilanteita, joissa kaikille organisaatioille ei välttämättä löydy jo valmiiksi sopivaa kokonaisarkkitehtuurikehystä, ja sopivan kehyksen suunnitteleminen voi olla paitsi haasteellista, myös aikaa vievää. Tämän kaltaisissa tilanteissa organisaation tulee kyetä selvittämään, olisiko kehyksen suunnitteleminen itse kannattavaa käytettävissä olevien resurssien valossa. Näin ollen on ensiarvoisen tärkeää käydä tarkkaan läpi jo olemassa olevat kehykset, jotta organisaatio ei törmää luvun alussa käsiteltyihin ongelmiin.

Muita aihetta koskevassa tutkimuksessa esille nousseita haasteita oli muun muuassa kokonaisarkkitehtuurin ylläpidon mahdollinen hankaluus. Vaikka teknologia ei olekaan pääosassa kokonaisarkkitehtuurissa, tietojärjestelmien mahdollinen nopea kehittyminen voi kuitenkin hankaloittaa kokonaisuuden ylläpitoa. Osa sovelluksista saattaa vanheta nopeastikin käyttökelvottomiksi IT-alan jatkuvasti muuttuvassa maailmassa. Näiden sovellusten korvaaminen saattaa koitua ongelmalliseksi, varsinkin jos kyseessä on organisaation kannalta kriittinen sovellus. Tämän ongelman ratkaisemiseksi on tärkeää, että organisaation tietohallinto antaa tukensa sovelluksen kehittämisestä tai vaihtamisesta vastaavalle taholle. [Kaisler *et al.* 2005, 5.] Tämä on myös osaltaan ristiriidassa kokonaisarkkitehtuurin hyötyjen kanssa. Käsitän tämän ongelman edelleen täsmentävän organisaation tietohallinnon vastuuta IT-palveluiden toiminnoista ja kokonaisarkkitehtuurin käyttämisessä.

Shah ja El Kourdi [2007, 40] ovat todenneet, että myös kokonaisarkkitehtuurin suunnitteluvaiheessa on omat ongelmansa, sillä se vaatii organisaation koosta riippuen suuret määrät tietoa. Tässä vaiheessa törmätään dokumentoinnin tarpeeseen ja hankaluuteen organisaation työntekijöiden erilaisissa työskentelytavoissa. Mikäli organisaatiolla ei ole jo olemassa standardiksi nousseita työkaluja ja -tapoja dokumentoinnin suhteen, niin tämä saattaa vaikeuttaa eri osastojen tietojen analysointia kokonaisarkkitehtuuria kehittävien suunnittelijoiden toimesta.

3. Tietojärjestelmät

Valtiovarainministeriön ylläpitämät JHS-suositukset sisältävät myös sanakirjan. Kyseisessä sanakirjassa termi 'tietojärjestelmä' määritellään seuraavasti: "Tietojärjestelmän muodostavat tiedot ja niiden käsittelysäännöt, käsittelyn henkilö- ja laiteresurssit sekä tiedonsiirtolaitteet ja toimintaohjeet. Tietojärjestelmäksi kutsutaan usein myös sovelluskokonaisuuksia." [JHS-sanasto 2016.]

Tämä siis tarkoittaa esimerkiksi organisaation eri toimijoiden - kuten lääkäreiden, hoitajien - ammattiryhmien sääntöjen ja potilastietojärjestelmän yhteistoimintaa. Potilas käy lääkärin vastaanotolla, jonka jälkeen lääkäri kirjoittaa havaintonsa tai diagnoosinsa käytettyyn potilastietojärjestelmään. Lääkäri voi myös mahdollisesti kirjoittaa lähetteen röntgeniin sairaalaan toiselle osastolle. Röntgen-osastolla voidaan käyttää jotain toista sovellusta, johon potilaan tiedot viedään. Tämä tiedonsiirto sovelluksesta toiseen ja toisen osaston työntekijälle halutaan toteuttaa mahdollisimman helposti ja ilman informaation katoamista. Tavoitteeseen pääsemistä edesauttaa se, että työntekijällä on mahdollisimman pieni rooli informaation siirtämisessä toiseen järjestelmään. Tiedonsiirto ei siis saisi tapahtua manuaalisesti viemällä tulostettu asiakirja toiselle osastolle, vaan kyseisen informaation tulisi siirtyä kokonaan digitaalisesti käytettäväksi; käytännössä siis mahdollisimman pienillä työntekijöiden toimilla inhimillisten virheiden vähentämiseksi.

3.1. Tietojärjestelmien haasteita

Kokonaisarkkitehtuurikehyksiä on olemassa monia erilaisia, tarpeesta riippuen. Esimerkiksi terveydenhoitoalalla on erittäin kriittisiä tietojärjestelmiä. Käytin esimerkkinä terveydenhoitoalaa, sillä siellä tietojärjestelmän toimimattomuus voi pahimmillaan johtaa jopa potilasturvallisuuden vakavaan vaarantumiseen ja sitä kautta ihmishenkien menettämiseen. Jokaisella organisaatiolla on kuitenkin omat määritelmänsä heidän kannaltaan kriittisille tietojärjestelmille tai sovelluksille. Nämä kriittiset järjestelmät luovatkin omia haasteita kokonaisarkkitehtuurin kehitykselle.

Grilo ja muut [2009, 297] toteavat, että yksi tärkeä osa esimerkiksi juuri sairaalan tietojärjestelmiä on niiden saumaton yhteentoimivuus muiden järjestelmien, prosessien, työntekijöiden ja järjestelmän muiden komponenttien kanssa. Tässä kohdassa jo pelkästään monta eri sovellusta voi aiheuttaa ongelmia, sillä usein varsinkin sairaalaympäristössä on sovelluksia monelta eri toimijalta moneen eri tarkoitukseen. Lähtökohtaisesti näitä sovelluksia ei ole rakennettu toimimaan keskenään saumattomasti, mutta nykypäivänä tähän tarkoitukseen on tehty erilaisia liittymiä. Kyseiset sovellukset sisältävätkin monenlaista dataa,

erilaisin parametrein, eivätkä ne välttämättä oletuksena sisällä sopivia tapoja edes tarvittavan tiedon viemiseen sovelluksesta toiseen.

Purnawan ja Surendo [2016, 1] ovat tulleet siihen lopputulokseen, että sairaalat eivät pidä yhteentoimivuutta tärkeänä tekijänä tietojärjestelmien kehitysvaiheessa. Tämä onkin heidän mukaansa yksi syy suurelle osalle sairaaloiden IT-projektien epäonnistumiseen [emt]. Näkemykseni mukaan tämänkaltaisen toiminnallisuuden mukaan saaminen jälkikäteen, vaikkapa yhteen tietojärjestelmän sisällä olevista sovelluksista, voi olla jopa mahdotonta annettujen resurssien puitteissa.

Ehkäpä kaikista suurimpana haasteena tässäkin voidaan pitää ihmistä. Varsinkin kokonaisarkkitehtuurin suunnitteluvaiheessa olisi erityisen tärkeää kuulla myös loppukäyttäjää [Ahmadi *et al.* 2014, 119]. Hankalan tästä asiasta voi tehdä se, että loppukäyttäjä ja asiantuntija eivät välttämättä ymmärrä toistensa ammattisanastoa. Toimittajan edustajana toimiva asiantuntija saattaa puhua liian teknisesti, ja toinen osapuoli taas käyttää liikaa oman alansa terminologiaa. Myös muutosvastarinta ja asenteet automaatioita kohtaan voivat aiheuttaa loppukäyttäjässä tai teknisessä asiantuntijassa, asenneongelmia. Yksi ulottuvuus asenneongelmassa voi olla se, että työntekijät itse pelkäävät oman työpaikansa menettämisestä uusien tietojärjestelmien kehittämisen seurauksena. Tämä ei aina myöskään ole aina aiheeton pelko, varsinkaan niissä tapauksissa, missä organisaatio hakee suoria kustannussäästöjä. Oman työpaikan menettämisen pelko saattaakin pahimmassa tapauksessa aiheuttaa sen, että suunnitteluvaiheessa ei anneta tarpeeksi tärkeätä informaatiota tietojärjestelmän tarvitsemista prosesseista tai muista tiedoista. Näin ollen lopputuloksena voi olla, että käyttöön otetaan huonosti suunniteltu sovellus tai kokonainen tietojärjestelmä, joka aiheuttaa lähinnä vain ongelmia.

3.2. Tietojärjestelmien rooli kokonaisarkkitehtuurissa

Ymmärtääksemme paremmin tietojärjestelmän roolia osana kokonaisarkkitehtuuria, täytyy meidän käydä vielä läpi kokonaisarkkitehtuurin osat. Tuomola [2014, 16] on todennut, että kokonaisarkkitehtuuri on jaoteltu vähintäänkin kahdella eri tavalla. Esimerkiksi TOGAF (The Open Group Architecture Framework) jakaa kokonaisarkkitehtuurin kokonaisuuden viiteen osaan: *Liiketoiminta-, prosessi-, integrointi-, ohjelmisto- ja infrastruktuuriarkkitehtuuri*. Tässä tutkielmassa viittaan kuitenkin edelleen JHS 179:n mukaiseen jakoon toiminta-, tieto-, tietojärjestelmä- ja teknologia-arkkitehtuuriin.

Näistä toiminta-arkkitehtuuri viittaa organisaation liiketoiminta osaan. Tämä arkkitehtuurin osio sisältää muun muassa organisaation strategian, tuot-

teet, palvelut ja liiketoiminta prosessit. [JUHTA, 20; Wijegunaratne *et al.* 2014, 6.]

Tietoarkkitehtuuri sisältää loogiset ja fyysiset tietovarantojen rakenteet. Tämä tarkoittaa käytännössä tietokantoja ja sen sisältämiä tietoja. Myös näiden tietojen väliset suhteet ovat tärkeä osa tietokantoja ja tätä kautta tietoarkkitehtuuria. [JUHTA 2017, 18; Wijegunaratne *et al.* 2014, 7.]

Tietojärjestelmäarkkitehtuuriin kuuluu sovellusten rakenne ja niihin kuuluvat suhteet organisaation muihin osiin. Myös näiden eri sovellusten tarjoamat palvelut ja toiminnot ovat osa tietojärjestelmäarkkitehtuuria. [JUHTA 2017, 19; Wijegunaratne *et al.* 2014, 7.]

Teknologia-arkkitehtuuri kuvaa organisaation IT-infrastruktuurin, eli organisaation sovellusten ja tietojärjestelmien fyysiset osuudet. JHS-suositusten mukaan tähän sisältyy myös organisaation teknologiaan liittyvät standardit ja tekniset vaihtoehdot. [JUHTA 2017, 18; Wijegunaratne *et al.* 2014, 7.]

Näiden kokonaisarkkitehtuurin eri osien perusteella on jo nähtävissä, kuinka suuressa osassa tietojärjestelmät ovat. Tulkintani perusteella tietojärjestelmät liittyvätkin näihin kaikkiin neljään eri kokonaisarkkitehtuurin osaan. Tarkastellaan aluksi toiminta-arkkitehtuuria. Menemättä sen tarkemmin toiminta-arkkitehtuurin syvällisempään olemukseen, niin yksi sen keskeisistä osista ovat *tavoitteet* [Reynolds 2010, 9]. Kaikilla organisaatioilla on oma päätavoitteensa, ja yksi kokonaisarkkitehtuurin hyödyistä olikin edelleen se, että kaikki osastot tietävät organisaation liiketoiminnan tavoitteet. Sen varmistamiseen, että kaikki organisaation työntekijät tietävät nämä tavoitteet vaaditaan toimiva tietojärjestelmä, joka omalta osaltaan auttaa tavoitteiden saavuttamisen seurantaa. Yhtenä osana toiminta-arkkitehtuuria mainittiin myös prosessit, jotka ovat usein yhteydessä kirjaamiskäytäntöjen kautta tietojärjestelmiin. Esimerkiksi käy kommunikaatio kahden eri osaston sovellusten tai työntekijöiden välillä.

Tietoarkkitehtuurin kuvauksen perusteella vaikuttaisi siltä, että tietojärjestelmien rooli ei olisi suuri. Tämä arkkitehtuurin osa sisältää tietokantojen rakenteen, joita usein erilaiset tietojärjestelmät vaativat. Tämän ja aikaisemman kuvauksen perusteella tulkitsenkin, että tietoarkkitehtuurit ovat enemmänkin tärkeä osa tietojärjestelmien toimintaa, eikä toisin päin. JHS-suosituksissa tietoarkkitehtuurin suunnitteluosan läpikäyminen auttaa vielä lisää ymmärtämään tämän suhdetta tietojärjestelmiin: ”Tietoarkkitehtuurin suunnittelun tavoitteena on luoda organisaatiotasoinen yhteinen näkemys keskeisestä tietopääomasta sekä helpottaa tiedon löytämistä, välittämistä ja hallintaa. Suunnittelulla tähdätään tietorakenteiden vakiointiin ja sen mahdollistamaan tietojen uudelleenhyödynnettävyyteen.” [JUHTA 2017, 18.] Tulkitsen tämän kuvaavan osaltaan myös organisaation sisäisten tietojärjestelmien rakentamista.

Teknologia-arkkitehtuuri on olennainen osa tietojärjestelmiä. Tietojärjestelmien osana olevat sovellukset vaativat fyysisen palvelinalustan, jolla ne sijaitsevat. Teknologia-arkkitehtuuri myös määrittelee organisaation tekniset standardit, jotka taas joissain tapauksissa voivat vaikuttaa tiettyjen sovelluksien toimintaan [JUHTA 2017, 18]. Tätä kautta teknologia-arkkitehtuuri vaikuttaa osaltaan siihen minkälaisia lopullisista tietojärjestelmistä voi rakentua.

Tietojärjestelmäarkkitehtuurin rooli suhteessa tietojärjestelmiin on ilmeinen. Roolin sijaan tässä yhteydessä tarkastelen tietojärjestelmäarkkitehtuurin osalta hieman sen suunnitteluvaihetta. Tämä auttaa hahmottamaan tietojärjestelmäarkkitehtuurin kokonaisuutta liittyen muihin arkkitehtuureihin. Tao ja muut [2015, 628] toteavat aikaisemmin läpikäydyn toiminta-arkkitehtuurin olevan tärkeä osa tietojärjestelmäarkkitehtuurin suunnittelua. Tällä tarkoitetaan sitä, että liiketoiminta-arkkitehtuurin suunnittelusta saadut lopputulokset ideaalitapauksessa hyödynnetään tietojärjestelmäarkkitehtuurin rakentamisessa. Ideaalitapauksessa siksi, että tämä ei välttämättä ole aina mahdollista kaikkien organisaatiossa olevien tietojärjestelmien tai niihin kuuluvien sovellusten kanssa. Tämän syitä sivuttiin kohdassa 3.1. Tietojärjestelmäarkkitehtuurin suhdetta tieto- ja teknologia-arkkitehtuureihin käsittelin jo hieman aiemmin.

Tulkintani mukaan tietojärjestelmät vaikuttavatkin jollain tavalla jokaisen yllämainitun arkkitehtuurinäkökulman rakentamista ja toiminnallisuutta. Viimeistään tätä kautta myös *koko* organisaation toimintaa kokonaisarkkitehtuurin käyttöönottamisen alusta saakka.

4. Yhteenveto

Kokonaisarkkitehtuuri on abstrakti ja sisällöltään muuttuva ja osin vakiintumaton käsite, jonka takia siihen liittyvässä tutkimuksessa esiintyy paikoin ristiriitaisia tulkintoja. Mainitut ristiriidat saattavatkin aiheuttaa ongelmia sekä hyötyjen että ongelmien tutkimisessa. Eräs aihetta koskevassa tutkimuksessa toistuvasti esiin nouseva hyöty tai tavoite on kuitenkin kiteytettävissä toteamukseen *”Yksinkertaisempaa IT:tä”*. Tämä ajatus mielestäni onnistuneesti summaa koko kokonaisarkkitehtuurin. Tehdään organisaation IT-palveluista ja -prosesseista yksinkertaisempia ja näin ollen myös loppukäyttäjille helpompia ymmärtää.

Kokonaisarkkitehtuurin rakentamiseen vaadittavia kehyksiä on luotu jo lukemattomia, sillä kokonaisarkkitehtuuri rakennetaan aina organisaation omien tavoitteiden tai toiveiden mukaisesti. Voimmekin sanoa, että se mikä osa luodaan tai mitä kokonaisarkkitehtuurin osaa päivitetään eri organisaatiossa, vaihtelee. Voidaankin sanoa, että jokainen rakennettu kokonaisarkkitehtuuri on aina uniikki kokonaisuus.

Myös tietojärjestelmän kohdalla on tärkeää huomata, että kyseessä on melko laaja käsite. Suppeimmillaan se voi käsittää pelkät organisaation tietokannat, kun taas laajimmillaan siihen saattavat sisältyä jopa kaikki organisaation resurssit. Onkin siis selvää, että tapauksessa, jossa on iso organisaatio kyseessä, vaaditaan toimivaa kokonaisarkkitehtuuria pitämään organisaation toimintoja kasassa.

Tietojärjestelmien ja kokonaisarkkitehtuurien kehittämisen kannalta keskeisenä tekijänä nousi toistuvasti esille kommunikaatio. Erilaiset kommunikaatioongelmat kehittäjien ja käyttäjien välillä voivatkin aiheuttaa ongelmia lopputuotteessa. Tästä syystä näkisin, että käyttäjien kuunteleminen on ensiarvoisen tärkeää. Pelkkä kuuntelu ei myöskään yksinään riitä, vaan kehittäjien tulisi myös osata kysyä oikeat kysymykset. Tämän takia onkin tärkeää, että kehittäjät ymmärtävät asiakkaiden käyttämää ammattisanastoa ja heillä olisi mahdollisesti myös substanssiosaamista käsillä olevasta alasta.

Kokonaisarkkitehtuurin eri osien suhdetta tietojärjestelmien kanssa voidaan koostaa melko lyhyesti, sillä tietojärjestelmät liittyvät jollain tavalla jokaiseen JHS 179 -suositusten käyttämään kokonaisarkkitehtuurin osakokonaisuuksiin. Tietojärjestelmät liittyvätkin näihin neljään näkökulmaan sekä abstraktilla että fyysisillä tasoilla.

Kokonaisarkkitehtuuria on kuluvalla vuosituhannella tutkittu melko paljon. Tutkimus on kuitenkin osin rajoittunut koskemaan vain esimerkiksi suunnittelun vaiheita. Tämän tutkielman ja oman työkokemukseni perusteella erityisen kiinnostava jatkotutkimuksen aihe liittyisi siihen, miten yrityskaupat ja fuusioituminen vaikuttavat kokonaisarkkitehtuuriin. Tämänkaltaisissa tapauksissa on kuitenkin tyypillistä, että molemmilla yrityksillä on olemassa omat, mahdollisesti hyvin kriittisetkin, tietojärjestelmänsä. Olisikin syytä pohtia sitä, minkälaisia ongelmia tästä sulautuksesta voi seurata? Entä pystytäänkö niitä ratkaisemaan pelkästään hyvän kokonaisarkkitehtuurin avulla?

Viiteluettelo

Maryam Ahmadi, Shahla Damanabi and Farahnaz Sadoughi. 2014. A Comparative study of the proposed models for the components of the national health information system. *Acta Informatica Medica* 22, 2, 115-119.

Aurona Gerber, Alta van der Merwe and Kevin Bayes. 2013. An investigation into UML case tool support for the Zachman Framework. In: *Proc. of Enterprise Systems Conference*, 22-30.

Tom Graves. 2009. *Enterprise Architecture: a Pocket Guide.* IT Governance Publishing.

António Grilo, Ricardo Jardim-Goncalves, Luis Velez Lapao and Virgilio Cruz-Machado. 2009. Challenges for the development of interoperable information systems in healthcare organisations. In: *Proc. of 2009 International Conference on Interoperability for Enterprise Software and Applications China,* 297-301.

JHS-sanasto. Valtiovarainministeriö. Verkkolähde. Viitattu 18.12.2017. http://jhs-sanasto.jhs-suositukset.fi/JHS/fi/

JUHTA-Julkisen hallinnon tietohallinnon neuvottelukunta. 2017. JHS 179 Kokonaisarkkitehtuurin suunnittelu ja kehittäminen. Verkkolähde. Viitattu 18.12.2017. Saatavissa: http://docs.jhs-suositukset.fi/jhs-suositukset/JHS179/JHS179.pdf

Stephen H. Kaisler, Frank Armour and Michael Valivullah. 2005. Enterprise architecting: critical problems. In: *Proc. of the 37th Havaii International Conference on System Sciences.*

Juha Lemmetti and Samuli Pekkola. 2012. Understanding enterprise architecture: Perceptions by the Finnish public sector. In: *Proc. of Electronic Government: 11th IFIP WG 8.5 International Conference, EGOV 2012.* 162-173.

Eetu Niemi and Samuli Pekkola. 2009. Adapting the Delone and McLean model for the enterprise architecture benefit realization process. In: *Proc. of the 42dn Havaii International Conference on System Sciences,* 3585-3594.

Eetu Niemi and Samuli Pekkola 2013. Enterprise architecture qualitu attributes: A case study. In: *Proc. of the 46th Havaii International Conference on System Sciences,* 3878-3887.

Eetu Niemi and Samuli Pekkola. 2017. Using enterprise architecture artefacts in an organization. *Enterprise Information Systems* 11, 3, 313-338.

Dilla Anindita Purnawan and Kridanto Surendro. 2016. Building enterprise architecture for hospital information system. In: *Proc. of the Fourth International Conference on Information and Communication Technologies.* 185-190.

Toomas Tamm, Peter B. Seddon, Graeme Shanks and Peter Reynolds. 2011. How does enterprise architecture add value to organisations. *Communications of the Association for Information Systems.* 28, Article 10, 29 pages.

Chris Reynolds. 2010. Introduction to Business Architecture. Course PTR.

Jaap Schekkerman. 2004, second edition. *How to Survive in the Jungle of Enterprise Architecture Frameworks.* Trafford Publishing.

Hanifa Shah and Mohamed El Kourdi. 2007. Frameworks for enterprise architecture. *IT Professional* 9, 5, 36-41.

Zhi-Gang Tao, Yun-Feng Luo, Chang-Xin Chen, Ming-Zhe Wang and Feng Ni. 2015. Enterprise application architecture development based on DoDaf and TOGAF. *Enterprise Information Systems* 11, 5, 627-651.

Juuso Tuomola. 2014. *Kokonaisarkkitehtuurityön saattaminen osaksi toiminnan kehittämistä ja johtamista: Terveyden ja hyvinvoinnin kohdealue.* Diplomityö. Talouden ja rakentamisen tiedekunta, Tampereen teknillinen yliopisto.

Juuso Tuomola. 2017. Luento Tampereen yliopiston Tietotekniikka ja yhteiskunta -jaksolla kevätlukukaudella 2017.

John A. Zachman. 1987. A framework for information systems architecture. *IBM Systems Journal* 26, 3, 276-292.

Inji Wijegunaratne, George Fernandez and Peter Evans-Greenwood. 2014. *Enterprise Architecture for Business Success*. Bentham Science Publishers.

Funktionaalisesta ohjelmoinnista

Enna Raerinne

Tiivistelmä.
Funktionaalista ohjelmointia on tutkittu jo useita vuosikymmeniä, mutta tämän ohjelmointiparadigman laajempi hyödyntäminen ohjelmistoteollisuudessa ja ohjelmoinnin opetuksessa on alkanut vasta hiljattain. Tässä tutkielmassa perehdytään funktionaaliseen ohjelmointiin sille ominaisten piirteiden ja lambdakalkyylin kautta. Funktionaalisista ohjelmointikielistä tarkastellaan erityisesti Haskellia. Tutkielmassa pohditaan funktionaalisen ohjelmoinnin hyötyjä samanaikaisuuden ja rinnakkaisuuden toteutuksessa sekä kerrotaan funktionaalisten kielten käytöstä ohjelmoinnin opetuksen johdantokursseilla.

Avainsanat ja -sanonnat: Funktionaalinen ohjelmointi, lambdakalkyyli, Haskell, ohjelmoinnin opetus.

1. Johdanto

Funktionaalinen ohjelmointi on vuosikymmeniä tutkittu ohjelmointiparadigma, jota on aloitettu vasta nykyään käyttämään ohjelmistoteollisuudessa [Hinsen 2009]. Esimerkiksi Haskell-ohjelmointikielellä on tehty monenlaisia kaupallisia ohjelmistoja tai sen osia. Sitä käytetään niin ilmailu-, puolustus- ja talousalalla kuin laitteistosuunnitteluyrityksissäkin. [HaskellWiki 2017]

Funktionaalista ohjelmointia on aloitettu hiljattain hyödyntämään myös ohjelmoinnin opetuksen johdantokursseilla [Choppella *et al.* 2012]. Ohjelmoinnin opetus on erittäin tärkeää modernissa yhteiskunnassa, sillä opetus vaikuttaa ohjelmoijan kehittymiseen työssään. Ensimmäinen ohjelmointikieli vaikuttaa ohjelmoijan ajattelutapaan, mikä näkyy ohjelmointityylissä ja -tekniikoissa sekä kirjoitetun koodin laadussa. [Vujošević-Janičić and Tošić 2008]

Tässä tutkielmassa perehdytään funktionaalisen ohjelmoinnin yleisiin piirteisiin ja hyötyihin sekä funktionaalisiin ohjelmointikieliin ja niiden käyttöön ohjelmoinnin opetuksessa. Ensin käsitellään lambdakalkyyliä, johon funktionaalinen ohjelmointiparadigma perustuu. Tarkoitus on valottaa lambdakalkyylin ideaa kertomalla sen määritelmästä, lausekkeiden evaluoinnista ja yleisesti laskennasta.

Lambdakalkyylin jälkeen käsitellään funktionaaliselle ohjelmoinnille ominaisia piirteitä kuten funktioita, sivuvaikutuksettomuutta, rekursion käyttöä sekä erilaisia evaluointitapoja, erityisesti laiskaa evaluointia. Kaikissa funktio-

naalisissa kielissä ei välttämättä ole näitä ominaisuuksia sellaisina kuin ne tutkielmassa esitellään. Myös tietorakenteita käsitellään lyhyesti, sillä edellä mainitut ominaisuudet vaikuttavat niihin. Tässä tutkielmassa kerrotaan myös pääpiirteittäin funktionaalisen ohjelmoinnin hyödyistä samanaikaisuuden ja rinnakkaisuuden toteutuksessa.

Funktionaalisia ohjelmointikieliä tarkastellaan yleisesti kertomalla niiden ominaisuuksista ja siitä, miten ne ovat vaikuttaneet funktionaalisen ohjelmointiparadigman kehitykseen. Haskellista kerrotaan muita kieliä laajemmin ja sitä käytetään esimerkeissä, koska se yhdistää aiemmin kehitettyjen kielten piirteitä ja siinä on edellä esitellyt yleiset funktionaalisten kielten ominaisuudet. Kaikkia Haskell-kielen lukuisia ominaisuuksia ei käsitellä, vaan keskitytään tyypitykseen, funktioiden rakenteeseen ja syntaksiin sekä listojen käyttöön tietorakenteena.

Viimeiseksi tarkastellaan funktionaalisten kielten soveltumista ohjelmoinnin johdantokurssien opetuskieliksi eri näkökulmien ja ohjelmoinnin opetuksen tavoitteiden kautta.

2. Lambdakalkyyli

Funktionaalinen ohjelmointi perustuu Alonzo Churchin 1930-luvulla kehittämään *lambdakalkyylinä* (lambda calculus) tunnettuun laskentamalliin [Michaelson 1989]. Lambdakalkyyli muodostaa perustan funktionaalisen ohjelmoinnin keskeisille piirteille [Vichare 2013]. Tämä formaali malli koostuu yksinkertaisesta joukosta primitiivioperaatioita ja niiden käyttöä ohjaavista säännöistä [Michaelson 1989]. Lambdakalkyyli on yksinkertaisuudessaan erittäin ilmaisuvoimainen [Vichare 2013]. Sitä voidaan käyttää ohjelmointikielien kuvaamiseen [Michaelson 1989] ja niiden matemaattiseen analysointiin [Vichare 2013]. Lambdakalkyyli voidaan käsittää itsekin ohjelmointikieleksi [Rojas 2015].

Lambdakalkyyli on kehitetty samaan aikaan kuin Turingin kone. Ne ovat erillisiä malleja, mutta ne on todistettu ekvivalenteiksi eli niillä pystytään muun muassa mallintamaan toisiaan. Turingin kone on malli, jossa keskitytään mekanisoituun symbolin manipulointiin, joka perustuu sijoitukseen ja evaluointiin aikajärjestyksessä. Tämä on merkittävä ero lambdakalkyyliin, joka painottaa laskennan olevan rakenteista funktion soveltamista, jossa evaluointijärjestyksellä ei ole merkitystä. [Michaelson 1989] Lambdakalkyylissä ei välitetä teknisestä toteutuksesta ja se onkin lähempänä ohjelmistoa [Rojas 2015]. Tästä syystä funktionaalisen ohjelmointitekniikan synty sai odottaa 1950-luvulle [Barendregt 1997, Hinsen 2009].

2.1. Määritelmä

Lambdakalkyyli on järjestelmä *lambdalausekkeiden* käsittelyyn. Lauseke voi olla *nimi*, *funktio* tai *applikaatio*. Sama *Backus–Naur-muodossa* ilmaistuna [Michaelson 1989]:

```
<lauseke> ::= <nimi>|<funktio>|<applikaatio>.        (1)
```

Nimi vastaa muuttujaa eli se on merkkijono jonkin objektin tunnistamiseen. Funktio on tässä kuin matemaattinen funktio. Se on sääntö, jonka mukaan yhden joukon objekti kuvautuu toiseen objektiin. [Vichare 2013] Voidaan sanoa myös, että funktio kuvaa lähtöjoukon arvon maalijoukon arvoon. Tärkeä piirre on, että funktio antaa aina saman tuloksen samalla syötteellä. [Hinsen 2009]

Lambdafunktio ilmaistaan käyttäen λ-symbolia ja pistettä. Se koostuu λ-symbolin jälkeisestä nimestä, jota seuraa piste ja lauseke. Sama Backus–Naur-muodossa [Michaelson 1989]:

```
<funktio> ::= "λ"<nimi>"."<lauseke>.                 (2)
```

Pisteen jälkeistä osiota voidaan kutsua myös funktion *rungoksi* (body). Koska funktion runkona on lambdalauseke, sen paikalla voi olla nimi (muuttuja), funktio tai applikaatio kuten lausekkeen säännöstä (1) näkyy. [Michaelson 1989] Yksinkertainen esimerkki lambdafunktiosta on *identiteettifunktio*

```
λ<nimi>.<nimi>,                                      (3)
```

joka on

```
λx.x,                                                (4)
```

kun nimenä käytetään pientä x-kirjainta. Funktio voidaan kirjoittaa selkeyden vuoksi myös kaarisulkeiden sisään. [Rojas 2015]

Nimi on funktion *sidottu muuttuja* (bound variable), jos funktion rungossa esiintyvä nimi (muuttuja) esiintyy myös funktion *päässä*, joksi voidaan kutsua pistettä edeltävää osaa. Esimerkiksi identiteettifunktiossa (4) *x* on sidottu muuttuja. Jos nimi ei esiinny funktion päässä, on kyseessä *vapaa muuttuja* (free variable). [Michaelson 1989, Rojas 2015]

Lambdalausekkeiden evaluoinnin ydinoperaatio, *applikaatio*, koostuu kahdesta lambdalausekkeesta. Sen Backus–Naur-muoto on

```
<applikaatio> ::= "("<lauseke>" "<lauseke>")".       (5)
```

Sulkeita ja lausekkeiden välistä väliä on hyvä käyttää selkeyden vuoksi. Se ei ole kuitenkaan pakollista kuten kuvasta 1 näkyy. Applikaatiosta yksinkertainen esimerkki on identiteettifunktion (4) applikointi eli soveltaminen tai käyttäminen itseensä

```
(λx.x λx.x),                                         (6)
```

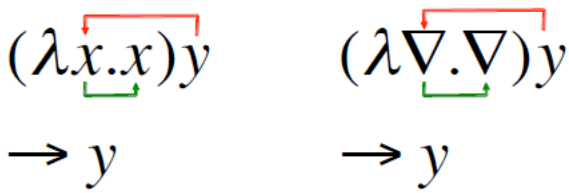
josta saadaan tulokseksi identiteettifunktio itse. Identiteettifunktio nimittäin antaa tulokseksi sellaisenaan saamansa argumentin, joka on tässä myös identiteettifunktio. [Michaelson 1989] Applikaatiota käsitellään tarkemmin lambdalausekkeiden evaluoinnin yhteydessä.

2.2. Lausekkeiden evaluointi

Applikaation kaksi lauseketta voidaan nimetä *funktiolausekkeeksi* (function expression) ja *argumenttilausekkeeksi* (argument expression) edellä mainitussa järjestyksessä [Michaelson 1989]. Esimerkiksi applikaatiossa

```
((λx.x) y)                                                    (7)
```

funktiolauseke on identiteettifunktio ja argumenttilauseke on muuttuja y. Applikaatiosta saadaan tulos, eli se evaluoidaan, korvaamalla funktion sidottu muuttuja argumenttilausekkeella. Tässä tapauksessa siis korvataan muuttuja x muuttujalla y. Koska identiteettifunktio palauttaa argumentin sellaisenaan, tulokseksi saadaan y. [Rojas 2015] Evaluoinnissa funktion pää ikään kuin häviää sidotun muuttujan korvaamisen jälkeen. Jäljelle jää runko, jossa sidottu muuttuja on korvattu argumenttilausekkeella. Tätä evaluointitapaa kutsutaan *β-säännöksi* tai myös *β-reduktioksi* (β-reduction) [Vichare 2013]. Sama asia on esitetty vielä kuvassa 1.



Kuva 1. Evaluointi kahdella eri nimisellä sidotulla muuttujalla x ja ∇. Nähdään, että funktion sidottu muuttuja on vain paikan pitäjänä. [Rojas 2015]

Joskus sidotun muuttujan ja vapaan muuttujan nimi voi sattua samaksi. Tällöin käytetään *α-konversiota* (α-conversion) tai yksinkertaisemmin *α-sääntöä*. Siinä muutetaan sidotun muuttujan nimi, jotta konfliktia ei tulisi nimien kanssa. [Michaelson 1989] Kuten kuvassa 1 nähdään, sidotun muuttujan nimellä ei ole merkitystä ja se voidaan vaihtaa muuttamatta funktion toiminnallisuutta. Esimerkiksi identiteettifunktio voidaan ilmaista myös seuraavanlaisesti:

```
λkissa.kissa.                                                 (8)
```

Lambda-lausekkeita voidaan yksinkertaistaa *η-reduktion* (η-reduction) avulla. Lauseke, joka on muotoa

```
λ<nimi>.(<lauseke> <nimi>),                                   (9)
```

saadaan *η-säännöllä* muotoon

```
<lauseke>,                                                    (10)
```

koska kun lauseke (9) saa minkä tahansa lausekkeen argumentiksi, tulos (10) on lausekkeen ja argumentin applikaatio [Michaelson 1989]:

```
(λ<nimi>.(<lauseke> <nimi>) <argumentti>) =>
(<lauseke> <argumentti>).                                     (11)
```

Lambdafunktio voi ottaa vastaan kaksi argumenttia, jos sen runkona on toinen lambdafunktio. Jotta lambdafunktiolle voisi antaa *n* argumenttia, tulee runkona olla funktio, jonka runkona on taas funktio ja niin edelleen, kunnes funktion päitä on peräkkäin *n* kappaletta. Esimerkiksi funktiossa

```
λx.λy.(x y)                                                   (12)
```

on sidottu muuttuja *x* ja funktion (12) runkona on toinen lambdafunktio

```
λy.(x y),                                                     (13)
```

jolla on sidottu muuttuja *y* ja sen rungon lambdalauseke on applikaatio

```
(x y).                                                        (14)
```

Kun koko funktiolle (12) antaa argumentteina funktion

```
λs.(s s),                                                     (15)
```

joka applikoi saamansa argumentin argumenttiin itseensä ja identiteettifunktion (4), niin saadaan

```
((λx.λy.(x y) λs.(s s)) λx.x) =>
(λy.(λs.(s s) y) λx.x) =>
(λs.(s s) λx.x) => (λx.x λx.x) => λx.x.                       (16)
```

Funktio (12) applikoidaan ensimmäiseen argumenttiin (15) palauttaen uuden funktion, jossa sidottu muuttuja *x* on korvattu funktiolla (15). Tämä uusi funktio applikoidaan seuraavan argumentin eli identiteettifunktion kanssa korvaamalla sidottu muuttuja *y* identiteettifunktiolla. Tulokseksi saadaan funktion (15) ja identiteettifunktion applikaatio, josta tulee identiteettifunktion applikointi itseensä. Tästä lopputuloksena on identiteettifunktio itse, koska sidottu muuttuja *s* korvataan identiteettifunktiolla ja sen jälkeen oikeanpuoleisella identiteettifunktiolla korvataan vasemmanpuoleisen identiteettifunktion sidottu muuttuja *x*. [Michaelson 1989]

Lambdakalkyylissä ei ole varmaa, että evaluointi päättyy. Esimerkiksi, jos funktion (15) applikoi itseensä, applikointi ei koskaan pääty.

```
(λs.(s s) λs.(s s)) => (λs.(s s) λs.(s s)) => ...             (17)
```

Tätä voidaan hyödyntää lambdakalkyylissä *rekursiivisia funktioita* luodessa. [Michaelson 1989]

2.3. Laskenta

Lambdakalkyylissä kaikki ilmaistaan funktiolla, jopa numerot ja totuusarvot. Esimerkiksi luonnolliset luvut saadaan muodostettua, kun keskitytään luvun toiminnallisuuteen eli laskemiseen. [Vichare 2013] Luku saadaan siis funktiosta, joka esittää nollaa ja luvun määrän verran jotain funktiota *f* applikoituna nollaan [Rojas 2015]. Nollaa esittää funktio

```
λf.λx.x,                                                    (18)
```

jossa ei vielä ole yhtään funktion *f* applikointia. Luvut 1, 2, ..., *n* esitetään seuraavasti funktiolla (18) ja jollain funktiolla *f*:

```
1:    λf.λx.(f x)
2:    λf.λx.(f (f x))
3:    λf.λx.(f (f (f x)))
...
n:    λf.λx.(f ... (f x)...).                               (19)
```

Näitä lukuja kutsutaan *Churchin numeraaleiksi* Alonzo Churchin mukaan. Näin ilmaistuna luvut itsensä voidaan ajatella aktiiviseksi dataksi eikä passiiviseksi kuten yleensä on tapana. [Vichare 2013] Luvuilla voidaan tietenkin tehdä laskuja vertailuoperaatioita, kuten luvuilla yleensä, ja kun ne on määritelty tällä tapaa, niistä on hyötyä aritmetiikan ulkopuolellakin [Rojas 2015].

Aritmetiikan, Boolen algebran, listojen ja parien toteutuksessa tarvitaan kaksi tärkeää funktiota, joista ensimmäinen valitsee ensimmäisenä saamansa argumentin ja toinen toisena saamansa argumentin. Ensimmäinen funktio on seuraavanlainen:

```
λeka.λtoka.eka.                                             (20)
```

Nimet *eka* ja *toka* on valittu selventämään järjestystä. Kun funktio (20) saa kaksi argumenttia, se evaluoituu ensimmäiseksi argumentiksi, joka voi olla mikä tahansa lambdalauseke:

```
((λeka.λtoka.eka argumentti_1) argumentti_2) =>
(λtoka.argumentti_1 argumentti_2) =>
argumentti_1.                                               (21)
```

Tässä toinen argumentti katoaa, koska sidottu muuttuja *toka* ei esiinny funktion rungossa. [Michaelson 1989]

Funktio, joka valitsee toisen argumenttinsa, on seuraavanlainen [Michaelson 1989]:

```
λeka.λtoka.toka.                                            (22)
```

Mielenkiintoista tässä funktiossa on se, että se muistuttaa funktiota (18), joka esittää nollaa Churchin numeraaleissa. Funktion applikointi kahteen argumenttiin näyttää seuraavanlaiselta [Michaelson 1989]:

```
((λeka.λtoka.toka argumentti_1) argumentti_2) =>
(λtoka.toka argumentti_2) =>
argumentti_2.                                    (23)
```

Pari (*a*, *b*) koostuu kahdesta objektista *a* ja *b*, jotka on prosessointia varten niin sanotusti liitetty yhteen eli niitä halutaan käsitellä yhdessä. Lambdakalkyylissä pari voidaan muodostaa seuraavalla tavalla [Vichare 2013]:

```
(a, b) : λa.λb.λf.((f a) b).                     (24)
```

Parifunktiossa sidottu muuttuja *f* on paikan pitäjä esimerkiksi funktiolle, joka käsittelee paria jotenkin [Michaelson 1989]. Esimerkiksi funktio (20), jolla voidaan valita parin ensimmäinen objekti tai funktio (22), jolla voidaan valita parin toinen objekti.

Ensin parifunktiolle täytyy kuitenkin antaa sen sisältämät objektit, jotka ovat lambdalausekkeita [Vichare 2013]. Ne voisivat olla esimerkiksi luvut 1 ja 2 (19), jotka voidaan pitää normaalissa muodossaan selkeyden vuoksi [Rojas 2015]:

```
((λa.λb.λf.((f a) b) 1) 2) =>
(λb.λf.((f 1) b) 2) =>
λf.((f 1) 2).                                    (25)
```

Lambdakalkyylissä evaluointia ei ole pakko viedä loppuun asti. Näin voidaan esimerkiksi muodostaa pelkkä pari objekteineen (25), kun sitä ei haluta käsitellä vielä. Tätä piirrettä kutsutaan *osittaiseksi evaluoinniksi* (partial evaluation), jota käytettiin myös applikaatiossa (16). Osittaisesta evaluoinnista seuraa, että lausekkeiden tulokset voivat olla funktioita, ja kuten edellä on nähty, funktiot voivat saada myös funktioita argumenteikseen. [Vichare 2013]

Parifunktio (25) voisi siis saada vielä yhden argumentin, joka käsittelee paria jotenkin. Jos annetaan esimerkiksi funktio (20), saadaan tulokseksi parin ensimmäisen objekti:

```
(λf.((f 1) 2) λeka.λtoka.eka) =>
((λeka.λtoka.eka 1) 2) =>
((λtoka.1) 2) => [1 lambdafunktioksi]
(λtoka.λf.λx.(f x) 2) =>
λf.λx.(f x).                                     (26)
```

Boolen algebra pystytään luomaan lambdakalkyylissä hyödyntämällä funktioita (20) ja (22). Totuusarvot tosi ja epätosi voidaan ajatella olevan valinnat

näistä kahdesta vaihtoehdosta. Näin funktiot (20) ja (22) voivat esittää totuusarvoja, jolloin myös tosi ja epätosi voidaan nyt nähdä aktiivisina tekijöinä niiden normaalin passiivisen roolin sijaan. [Vichare 2013]

Dataa pidetään normaalisti passiivisena. Lambdakalkyyli antaa sille aktiivisen roolin, kun data esitetään funktioina, kuten lukuina tai totuusarvoina. Koska kaikki esitetään funktioina, funktiot ovat sallittuja argumentteja toisille funktioille ja niitä voivat myös toiset funktiot palauttaa osittaisen evaluoinnin tuloksena. Evaluointijärjestyksellä ei ole merkitystä, sillä funktiot antavat aina saman tuloksen samoilla argumenteilla. Lambdakalkyyli näyttää, ettei arvon asetusta eli sijoitusta tarvita laskentaa varten, vaan *sitominen* (binding) riittää. Kun nimi on sidottu arvoon, arvo ei muutu. [Vichare 2013]

3. Funktionaalinen ohjelmointi

Funktionaalinen ohjelmointi on voimakas, puhdas ja matemaattisesti selkeä ohjelmointiparadigma, joka perustuu lambdakalkyyliin [Vichare 2013]. Se on vaihtoehtoinen lähestymistapa, joka ei muistuta tietokoneen toimintatapaa laitteiston tasolla, kuten perinteinen imperatiivinen ohjelmointiparadigma [Hinsen 2009]. Funktionaalinen ohjelmointi on deklaratiivista ohjelmointia, jossa ohjelma kertoo luonnollisella tavalla mitä se laskee. Siinä algoritmit voidaan kuvata muodossa, joka suoraan heijastaa algoritmin rakennetta. [Mukund 2007a] Tästä syystä funktionaalinen ohjelmointi on hyvin erilaista kuin imperatiivinen ohjelmointi ja sillä on omat piirteensä [Hinsen 2009].

3.1. Funktiot

Funktionaaliset ohjelmat koostuvat kokoelmasta funktioita [Mukund 2007a]. Funktionaalisessa ohjelmoinnissa funktiokutsut ovat siis ohjelmoinnin päärakenne [Michaelson 1989]. Funktiot ovat matemaattisia funktioita kuten lambdakalkyylissä eivätkä vain aliohjelmia, kuten yleensä imperatiivisissa kielissä, jotka palauttavat jonkin arvon tai muuttavat globaalia rakennetta. Funktionaalisen ohjelmoinnin funktiot antavat aina saman tuloksen samalla syötteellä kuten matemaattiset vastineensa. [Hinsen 2009]

Funktionaalisissa kielissä funktioita kohdellaan kuten arvoja [Michaelson 1989]. Samoin kuin lambdakalkyylissä, niitä voidaan antaa toisille funktioille argumentteina ja funktiot voivat palauttaa funktioita. Sekä *dataa* (arvot) että *prosesseja* (funktiot) voidaan kohdella samalla tavoin. Tästä johtuen ei välttämättä tarvita erillisiä käsitteitä *tietorakenteille* ja *koodin rakenteille*. [Vichare 2013]

Funktiota, joka palauttaa funktion tai saa funktion parametrinaan, kutsutaan *korkeamman kertaluvun funktioksi* (higher-order function). Kun taas funktiota, joka

käsittelee perinteistä dataa, kutsutaan *ensimmäisen kertaluvun funktioksi* (first-order function). [Hinsen 2009] Esimerkiksi korkeamman kertaluvun funktio *map* saa parametreina funktion ja listan ja käyttää funktiota jokaiseen alkioon erikseen. Tässä tapauksessa se siis lisää luvun kymmenen jokaiseen listan kokonaislukualkioon:

```
map (+10) [1,2,3] => [11,12,13].
```
(27)

Korkeamman kertaluvun funktiot voivat tehdä *osittaisia evaluointeja* kuten lambdakalkyylissä eli funktiolle voidaan antaa vähemmän argumentteja kuin mitä se odottaa saavansa [Vichare 2013]. Tuloksena on silloin funktio, jolle voidaan antaa loput argumentit myöhemmin. Funktiolle voidaan antaa siis kaikki tai osa argumenteista ja tulos on joko funktio tai arvo [Laufer and Thiruvathukal 2009]. Tätä kutsutaan englanninkielisellä termillä *currying* loogikko Haskell B. Curryn mukaan, joka kuvaili kyseisen tekniikan [Laufer and Thiruvathukal 2009].

Funktio *add* saa kaksi kokonaislukuparametria ja laskee ne yhteen:

```
add :: Int -> Int -> Int
add x y = x + y.
```
(28)

Jos funktiolle *add* antaa vain yhden parametrin kahden sijaan saadaan osittain evaluoitunut funktio. Kun funktiolle annetaan esimerkiksi parametrina luku 10, saadaan tuloksena funktio *add_ten*, joka lisää luvun kymmenen saamaansa kokonaislukuparametriin:

```
add_ten = add 10.
```
(29)

Kun funktiota *add_ten* kutsutaan parametrilla 3 saadaan tulokseksi luku 13:

```
add_ten 3 => 13.
```
(30)

3.2. Sivuvaikutuksettomuus

Funktiot eivät tee muuta kuin palauttavat arvon tai toisen funktion. Kuten matemaattiset vastineensa ne eivät muuta mitään globaalia rakennetta, sillä sellaisia ei edes ole kummassakaan. Ne eivät myöskään esimerkiksi kirjoita tiedostoon tai muuta muuttujan arvoa muistissa. Funktioilla ei siis ole *sivuvaikutuksia* toisin kuin imperatiivisessa ohjelmoinnissa. [Hinsen 2009]

Sivuvaikutuksettomuuden seurauksena suoritusjärjestyksellä ei ole merkitystä. Koska ei ole sivuvaikutuksia, jotka voisivat muuttaa funktion toimintaa, niin funktio voidaan suorittaa minä ajankohtana tahansa. Ohjelmoijan ei tarvitse siis huolehtia ohjelman suoritusjärjestyksen kuvaamisesta. [Hughes 1989]

Koska funktio voidaan evaluoida milloin tahansa, eikä sivuvaikutuksia ole, voi funktion tai lausekkeen korvata sen tuloksella ja päinvastoin, muuttamatta

ohjelman käytöstä [Laufer and Thiruvathukal 2009, Hughes 1989]. Tätä kutsutaan englanninkielisellä termillä *referential transparency* [Hughes 1989]. Tästä johtuen funktionaalisessa ohjelmoinnissa, kuten myös lambdakalkyylissä, funktiot antavat aina saman tuloksen samalla syötteellä [Hinsen 2009]. Referential transparency -ominaisuus helpottaa ohjelmien kirjoittamista ja virheiden etsimistä ja korjaamista [Hudak 1989].

Sivuvaikutuksia kuitenkin tarvitaan, kun ohjelman täytyy olla tekemisissä oikean maailman kanssa. Sivuvaikutukset tulee silloin rajoittaa tiettyihin ohjelman osiin, joita on mahdollisimman vähän. [Hinsen 2009] Funktioita, joilla ei ole sivuvaikutuksia, kutsutaan puhtaiksi. Toisaalta osa rakenteista on epäpuhtaita jo lähtökohtaisesti, kuten rakenteet *syötteelle* ja *tulosteelle*. [Laufer and Thiruvathukal 2009]

3.3. Rekursio

Funktionaalisessa ohjelmoinnissa muuttujia niiden perinteisessä merkityksessä ei tarvita, koska ohjelmat koostuvat vain funktioista, eivätkä funktiot muuta muuttujia [Hinsen 2009]. Niin sanottuja muuttujia ei käytetä kuin funktioiden formaaleina parametreina. Kun formaali argumentti on yhdistetty arvoonsa, se ei voi saada uutta arvoa. Funktioiden sisällä ei siis käytetä *arvon uudelleen asetusta* eli arvo ei muutu. [Michaelson 1989] Lambdakalkyylin tapaan nimi sidotaan arvoon eikä sitä voi enää muuttaa. Uutta arvoa varten tarvitaan aina uusi nimi.

Koska muuttujia ei käytetä imperatiivisen ohjelmoinnin muuttujien tapaan, käytössä ei ole toistorakenteita kuten perinteisiä for- ja while-silmukoita, joissa vaihdettaisiin muuttujan arvoa iteraatioiden välillä [Hinsen 2009]. Toisto esitetään rekursiivisilla funktiolla eli funktioilla, jotka kutsuvat itseään. Rekursiivisissa funktioissa tarvitaan ehto, jonka täyttyessä funktion suoritus lopulta päättyy. Kun ehto täyttyy, funktio palaa takaisin perättäisten kutsujen muodostamaa polkua, jossa se koostaa samalla lopullisen tuloksen. Esimerkkinä rekursiivisesta funktiosta voidaan mainita *reverse*, joka palauttaa listan, jossa alkioiden järjestys on käänteinen alkuperäiseen verrattuna [Mukund 2007a]. Funktio saa parametrina käännettävän listan:

```
reverse :: [a] -> [a]
reverse [] = []
reverse (x:xs) = reverse xs ++ [x].          (31)
```

Funktiossa *reverse* on niin sanottuna *reunaehtona* (edge condition) tyhjä lista. Kun funktiota kutsutaan tyhjän listan kanssa, funktio palauttaa tyhjän listan. Muuten funktio saa argumenttinaan listan ilman sen ensimmäistä alkiota, kun funktio kutsuu itseään uudelleen. Listan ensimmäisestä alkiosta x luodaan lista

ja se lisätään kutsun perään. Esimerkiksi, jos funktiolle *reverse* (31) annetaan parametrina lista, jossa ovat alkiot 2, 4, 6 ja 8, niin tulos muodostuu rekursiivisesti seuraavalla tavalla:

```
reverse [2,4,6,8]
=> reverse [4,6,8] ++ [2]
=> reverse [6,8] ++ [4] ++ [2]
=> reverse [8] ++ [6] ++ [4] ++ [2]
=> reverse [] ++ [8] ++ [6] ++ [4] ++ [2]
=> [] ++ [8] ++ [6] ++ [4] ++ [2]
=> [8] ++ [6] ++ [4] ++ [2]
=> [8, 6] ++ [4] ++ [2]
=> [8, 6, 4] ++ [2]
=> [8, 6, 4, 2].                                    (32)
```

3.4. Evaluointi

Applikatiivisessa järjestyksessä (applicative order) evaluointi tapahtuu siten, että parametrina annettava lauseke evaluoidaan ennen kuin se siirretään formaaliin parametriin [Hudak 1989; Michaelson 1989]. Lambdakalkyylissä applikatiivinen evaluointijärjestys kuitenkin tekee joissain tapauksissa turhaa työtä; esimerkiksi, jos evaluoitua lauseketta ei tarvitakaan. Se voi johtaa myös pahimmillaan päättymättömään evaluointiin lambdakalkyylissä. [Hudak 1989]

Normaali evaluointijärjestys (normal order), jota käytetään lambdakalkyylissä, toimii siten, että itse parametrilauseketta ei evaluoida ennen formaaliin parametriin sitomista [Hudak 1989; Michaelson 1989]. Jos normaalijärjestys on toteutettu naiivisti, tulee laskennasta tehotonta, sillä jokaisen formaalin parametrin esiintymän tulos evaluoidaan erikseen. Pahimmillaan lauseke, joka joudutaan evaluoimaan monta kertaa, on jo itsessään paljon aikaa vievä. Normaalijärjestys on lambdakalkyylissä oleellinen rekursion toteutuksessa. [Hudak 1989]

Funktionaalisessa ohjelmoinnissa on mahdollista viivästyttää monimutkaisen lausekkeen osien evaluointia, kunnes niitä tarvitaan lausekkeen lopullisen tuloksen saamiseksi. Tätä evaluointitapaa kutsutaan *laiskaksi evaluoinniksi* (lazy evaluation). [Laufer and Thiruvathukal 2009] Se yhdistää normaalin järjestyksen ja applikatiivisen järjestyksen hyödyt eli siis viivästyttää lausekkeiden evaluointia ja välttää saman lausekkeen evaluointia uudestaan. Täten laiskassa evaluoinnissa lauseke evaluoidaan silloin, kun sen arvoa tarvitaan korvaten lausekkeen kopiot evaluoinnin tuloksella. [Michaelson 1989]

Laiskassa evaluoinnissa lausekkeet evaluoidaan uloimmasta sisimpään, sillä sisemmät lausekkeet evaluoidaan vasta, kun niiden arvoja oikeasti tarvitaan [Mukund. 2007b]. Järjestys on siis suunnilleen vasemmalta oikealle [Barendregt 1997].

Laiskan evaluoinnin hyötynä on se, että voidaan käyttää äärettömiä rakenteita, esimerkiksi ääretöntä listaa [Laufer and Thiruvathukal 2009]. Tällainen on mahdollista, koska lista evaluoidaan vasta tarvittaessa. On nimittäin usein käsitteellisesti helpompi määrittää funktio palauttamaan tulos äärettömänä listana ja poimia tuloksesta tarvittava äärellinen osa. [Mukund 2007b]

Funktionaalisista kielistä osa on *laiskoja* eli ne käyttävät laiskaa evaluointitapaa. Toiset taas käyttävät applikatiivista järjestystä, jota käytettiin ensimmäisenä funktionaalisissa kielissä. [Barendregt 1997] Tämä johtui siitä, että normaali järjestys oli aikanaan vaikeaa toteuttaa tehokkaasti perinteisille tietokoneille [Hudak 1989].

3.5. Tietorakenteet

Funktionaalisessa ohjelmoinnissa tietorakenteiden osia ei pystytä muuttamaan yksi kerrallaan, koska arvoja ei voida sijoittaa. Tästä seuraa, että koko rakenne on kirjoitettava uudelleen muuttaen samalla eksplisiittisesti haluttua osaa. Funktionaalisissa kielissä käytetään imperatiivisesta ohjelmoinnista tuttujen taulukoiden sijaan listoja, koska koko taulukon kirjoittaminen auki yhden alkion vaihtamiseksi ei ole järkevää. [Michaelson 1989]

Listat ovat *sisäkkäisiä tietorakenteita* (nested data structures). Ne perustuvat rekursiivisiin notaatioihin, joissa operaatio koko rakenteelle kuvataan sen osiin kohdistuvilla rekursiivisilla operaatioilla. Sisäkkäiset tietorakenteet muistuttavat täten sisäkkäisten funktiokutsujen notaatiota. Tämän tyyppinen rakenteiden kuvaus tarjoaa vakioformaatin rakenteiden kuvaukselle, joka yksinkertaistaa virheiden etsimistä ja poistoa ja lopullista tulostamista, koska ei ole tarvetta kirjoittaa erityyppisille rakenteille erityisiä aliohjelmia tulostukseen tai syötteen lukuun tiedostosta. [Michaelson 1989]

Yksi syy globaalien rakenteiden puuttumiseen funktionaalisesta ohjelmoinnista on edellä mainittu sijoituksen puuttuminen, sillä jos globaali rakenne olisi olemassa niin sen osia ei pystyttäisi muuttamaan funktiosta käsin. Sen sijaan tietorakenteita siirretään kokonaisina funktiolta toiselle, joissa niitä voidaan muokata. Tästä syystä kyseiset tietorakenteet eivät ole samalla tavalla globaaleja kuin imperatiivisissa ohjelmissa, joissa aliohjelmat voivat suoraan muokata globaalia rakennetta. [Michaelson 1989]

Tietorakenteiden välityksestä seuraa, että funktiokutsut funktionaalisessa ohjelmassa ovat suurempia kuin vastineensa imperatiivisissa ohjelmissa. Hyötynä on kuitenkin se, että rakenteen käsittely funktioissa on aina huomioitava funktioiden määrityksissä ja kutsuissa. Tämä tekee datan kulun seuraamisesta helpompaa. [Michaelson 1989] Funktionaalisen ohjelman data muuntuu argumentista tulokseksi kulkiessaan läpi funktioiden, jotka muuntavat sitä kohti lopputulostaan [Mukund 2007a].

3.6. Hyödyt tietojenkäsittelyssä

Tietojenkäsittelyssä *samanaikaisuus* (concurrency) ja *rinnakkaisuus* (parallelism) ovat tulleet yhä tärkeämmiksi. Samanaikaisuudella tarkoitetaan saman datan käsittelyä useassa suoritussäikeessä, kun taas rinnakkaisuudella tarkoitetaan laskennallisen tehtävän jakamista useaan keskenään kommunikoivaan prosessiin, jotka ajetaan rinnan. Pääasiallinen syy edeltävien tekniikoiden käyttöön on se, ettei yksittäisen prosessorin teho enää parane samaa tahtia kuin ennen. Tästä seuraa, että tietokoneen tehokkuutta parannetaan lisäämällä prosessoreihin ytimiä. Tällaisten koneiden hyödyntämiseen tarvitaan samanaikaisuutta tai rinnakkaisuutta tai molempia tekniikoita. [Hinsen 2009]

Funktionaalista ohjelmointia pidetään lupaavana tekniikkana samanaikaisuuden ja rinnakkaisuuden ohjelmoinnissa, sillä nykyiset valtavirran tekniikat ovat alttiimpia virheille ja vaikeampia oppia. Puhdas funktionaalinen ohjelma ei sisällä muuttujia, jolloin ei tule ongelmia datan yhtenäisyyden kanssa eikä tarvita lukituksia. Funktionaalisessa ohjelmassa kaikki datariippuvuudet ovat eksplisiittisiä, mikä mahdollistaa suuren määrän muunnoksia ohjelmaan rinnakkaistamista varten niin, että ohjelman tulos ei muutu. [Hinsen 2009]

On kuitenkin huomattava, että funktionaalisten kielten kääntäjät eivät vielä pysty muuntamaan sarjallista ohjelmaa vastaavaksi rinnakkaiseksi ohjelmaksi, vaikka tämä teoriassa olisikin mahdollista. Ongelmaksi nousee sellaisen muunnoksen löytäminen, joka tekisi ohjelmasta kyllin tehokkaan annetulle rinnakkaiselle tietokoneelle ja syötteelle. Lähitulevaisuudelta voidaan odottaa kääntäjiä, jotka pystyvät tekemään rinnakkaisia ohjelmia puoliautomaattisesti ohjelmoijan antamien tehokkuusratkaisujen avulla. [Hinsen 2009]

4. Funktionaaliset ohjelmointikielet

Lambdakalkyyli on funktionaalisen ohjelmoinnin perusta ja siten myös funktionaalisten ohjelmointikielien pohja. Lambdakalkyyli voidaan ajatella funktionaaliseksi kieleksi. Sitä ei kuitenkaan ollut tarkoitettu sellaiseksi, sillä moderneja tietokoneitakaan ei vielä ollut, kun lambdakalkyyli kehitettiin. Modernit funktionaaliset kielet ovat periaatteessa kaunisteltua lambdakalkyyliä. [Hudak 1989]

4.1. Lisp-kieliperhe

Ensimmäisen Lisp-kielen kehitti John McCarthy tekoälytutkimusta varten 1950-luvulla. Nimi Lisp tulee sanoista *list processing* eli listan prosessointi. [Hinsen 2009] Vaikka lambdakalkyyliä pidetäänkin Lispin perustana, John McCarthy on itse todennut, että lambdakalkyylin rooli Lispin kehityksessä oli aika pieni. Lambdakalkyylin suurin vaikutus Lisp-kieleen oli *anonyymien funktioiden* määritys. [Hudak 1989]

Lisp-kielessä käytetään lambdakalkyylistäkin tuttuja korkeamman kertaluvun funktioita, mutta se ei ole puhtaasti funktionaalinen kieli. Lispissä on piirteitä, kuten sijoituslauseke, jotka aiheuttavat sivuvaikutuksia. Lisp käyttää pääasiassa applikatiivista evaluointijärjestystä. Ehtolausekkeissa kuitenkin käytetään normaalia evaluointijärjestystä eli viivytetään evaluointia. Näin rekursiivisten funktioiden määritys ei aiheuta ongelmia. Lisp-kieltä ei ole tyypitetty. [Hudak 1989]

Nimensä mukaisesti Lisp-kielessä käsitellään listoja. Siinä koodi- ja tietorakenteiden välillä ei ole sinänsä erottelua vaan koko Lisp-ohjelma on lista. [Michaelson 1989] Tästä seuraa, että Lisp-ohjelma voi manipuloida itseään ja toisia Lisp-ohjelmia [Michaelson 1989, Hinsen 2009]. Tämä ominaisuus on yhteinen koko Lisp-kieliperheelle. Lispin murteita ovat muun muassa Common Lisp, Scheme ja modernimpi ja alkuperäistä Lispiä puhtaampi Clojure. [Hinsen 2009]

4.2. Muita funktionaalisia kieliä

Lisp-kielen jälkeen seuraava merkittävä funktionaalinen kieli oli Peter Landinin 1960-luvulla kehittämä Iswim. Nimi tulee sanoista *if you see what I mean*. Landin arvosti lambdakalkyylin puhtautta ja ilmaisuvoimaa, mutta otti huomioon myös sen askeettisuuden kehittäessään Iswim-kieltä. Se esitteli uusia syntaktisia piirteitä kuten *let-* ja *where-* lauseet ja sisennyksen merkityksen lohkojen määrityksessä erottimien, kuten aaltosulkeiden, sijaan. [Hudak 1989]

FP oli yksi ensimmäisistä funktionaalisista kielistä, joka sai laaja-alaista huomiota. Sen kehitti 1970-luvulla John Backus, jonka merkittäviä töitä olivat muun muassa imperatiivisten kielien Fortranin ja ALGOLin ja Backus–Naur-muodon kehitys. Backusin työ imperatiivisen ohjelmoinnin parissa antoi painoarvoa hänen väitteelleen funktionaalisen ohjelmointiparadigman paremmuudesta imperatiiviseen verrattuna, erityisesti modernissa ohjelmistokehityksessä. Kuitenkin suurinta osaa FP-kielen piirteistä ei ole moderneissa funktionaalisissa kielissä, mutta se toi huomiota funktionaaliselle ohjelmointiparadigmalle, vaikka Backus itse teki eron hänen ehdottamansa mallin ja funktionaalisen ohjelmoinnin välillä. [Hudak 1989]

ML-kieliperhe on myös merkittävä funktionaalisten kielten ryhmä. Ensimmäinen ML-kieli kehitettiin 1970-luvulla alun perin rekursiivisia funktioita todistavan järjestelmän komentokieleksi. Kun se huomattiin itsessään mielenkiintoiseksi, sitä alettiin kehittää erillisenä funktionaalisena ohjelmointikielenä. ML ei ole puhtaasti funktionaalinen, vaan sisältää muuttujiin verrattavan rakenteen. Siinä ei ole myöskään *referential transparency* -ominaisuutta (kohta 3.2). Kuitenkin sen ohjelmointityyli on funktionaalinen. [Hudak 1989]

ML oli aikanaan yksi käytännöllisimmistä funktionaalisista kielistä. Siinä käytetään korkeamman kertaluvun funktioita ja siinä on yksinkertainen syötteen ja tulosteen käsittely, moduulit ja jopa poikkeukset. Merkittävin piirre ML-kielessä on kuitenkin sen tyyppijärjestelmä, joka näkyy vahvassa ja staattisessa tyypityksessä, lausekkeiden tyyppien päättelyssä ilman eksplisiittistä tyypitystä ja mahdollisuudessa määritellä monimuotoisia funktioita ja tietorakenteita. Monimuotoisen funktion argumentit voivat olla mitä tahansa tyyppiä, koska funktion toteutus on tyypeistä riippumaton. Sama koskee monimuotoisia tietorakenteita. ML-kielessä voidaan lisäksi käyttää käyttäjän määrittämiä konkreettisia ja abstrakteja tietorakenteita. [Hudak 1989] ML-kielissä funktiot määritetään hahmontunnistusta käyttämällä eli vertaamalla annettuja argumentteja määritettyihin argumenttivaihtoehtoihin. ML-kieliperheen jäseniä ovat muun muassa Standard ML, OCaml ja F#. [Hinsen 2009]

Uusia funktionaalisia kieliä ovat muun muassa Scala ja Nemerle, jotka lisäävät funktionaalisen ohjelmoinnin piirteitä olio-ohjelmointikieliin, mikä tekee näistä kielistä hybridikieliä. Esimerkiksi Scala on yhteensopiva Javan kanssa ja käännettyjä Scala-ohjelmia ajetaan Javan virtuaalikoneella. [Hinsen 2009] Imperatiivisiin ohjelmointikieliin on myös lisätty joitain pelkästään funktionaalisen ohjelmoinnin piirteitä kuten anonyymit lambdafunktiot Pythoniin.

4.3. Haskell

Haskell on funktionaalisen ohjelmointiyhteisön yhteistyön tuloksena syntynyt kieli, joka yhdistää aiemmin kehitettyjen funktionaalisten kielten piirteitä. Se kehitettiin 1980-luvulla ja on siitä lähtien ollut funktionaalisen ohjelmoinnin opetus- ja tutkimuskäytössä. Haskellin ensimmäinen standardi on vuodelta 1998 ja sitä on laajennettu lisää sen jälkeen. [Mukund 2007a]

Haskellilla kirjoitetut ohjelmat koostuvat funktioista, jotka muuntavat argumenttinsa tulokseksi. Haskellin funktiot rakentuvat kahdesta osasta eli funktion tyyppimäärittelystä ja itse funktion toteutuksesta. Funktion toteutus muodostuu säännöistä, jotka määrittelevät miten argumenteista saadaan funktion tulos. Esimerkiksi funktion *add* (28) tyyppimäärittely on

```
Int -> Int -> Int.                                    (33)
```

Funktiolla *add* on argumentti *x*, joka on tyyppiä *Int*. Funktio muodostaa osittaisella evaluoinnilla funktion *add x*, joka puolestaan saa *Int*-tyyppisen argumentin *y*. Tulokseksi saadaan argumenttien summa, joka on tyyppiä *Int* eli funktion tyyppimääritys on *currying*-prosessin (kohta 3.1) seurauksena itseasiassa muotoa

```
Int -> (Int -> Int).
```
(34)

Haskell-kielessä on siis pohjimmiltaan vain funktioita, jotka saavat yhden argumentin ja palauttavat tuloksen. Tyyppimääritys voidaan jättää myös pois, sillä Haskell osaa päätellä sen. [Mukund 2007a]

Haskell on samalla tavalla staattisesti ja monimuotoisesti tyypitetty kieli kuin ML [Hudak 1989]. Esimerkiksi *reverse*-funktio (31) on monimuotoisesti tyypitetty eli saa argumentikseen listan, jossa alkiot ovat määrittelemätöntä tyyppiä *a* ja funktio palauttaa saman tyyppisen listan. Listan alkioiden tyypillä ei ole väliä, koska funktio ei tee mitään tyyppiriippuvaista.

Tyyppimäärittelyä seuraa varsinainen funktion toteutus, joka yksinkertaisimmillaan koostuu yhdestä määrittelevästä yhtälöstä kuten esimerkiksi funktiossa *add* (28). Ensin tulee funktion nimi, jota seuraavat funktion argumentit välilyönneillä eroteltuna. Argumentteja seuraa yhtäsuuruusmerkki ja sen jälkeen jokin lauseke. Funktiossa *add* (28) yhtälön oikeanpuoleinen osa on aritmeettinen lauseke $x + y$. [Mukund 2007a]

Argumenttilistassa ei ole välttämätöntä sitoa arvoa nimeen, vaan se voidaan yrittää täsmätä tiettyyn hahmoon. Hahmo voi olla esimeriksi jokin tietty arvo. Tämä on mahdollista, koska Haskellin funktion toteutus voi sisältää useita erillisiä määrityksiä, jotka käydään läpi ylhäältä alaspäin. Jos kaikki argumentit täsmäävät yhteen määritykseen, niin valitaan se. Nimeen sidottu argumentti täsmää millä tahansa tyyppimäärityksen mukaisella argumentilla. Esimerkiksi looginen operaattori *xor* voidaan määrittää funktiona hahmontunnistusta hyödyntäen seuraavalla tavalla:

```
xor :: Bool -> Bool -> Bool
xor True False = True
xor False True = True
xor b1 b2 = False.
```
(35)

Jos funktion kutsu on *xor False True*, niin ensimmäinen määritys ei täsmää, mutta toinen täsmää ja saadaan tulos *True*. Jos taas funktion kutsu on *xor True True*, niin viimeinen määritys täsmää, koska siihen täsmäisi mikä tahansa yhdistelmä Boolen arvoja. [Mukund 2007a]

Kuten ohjelmointikielissä yleensä, Haskellissa käytetään myös ehtolausekkeita. Ehtolausekerakennetta kutsutaan Haskellissa englanninkielisellä termillä

guards eli vartijat. Nimi kuvaa ehtojen toimintaa, sillä ne rajoittavat pääsyä määritykseen, joka seuraa ehtoa. *Guards*-rakenne näyttää seuraavalta *xor*-funktiossa:

```
xor :: Bool -> Bool -> Bool
xor b1 b2 | b1 && not(b2) = True
          | not(b1) && b2 = True
          | otherwise     = False.
```
(36)

Ehdoissa ensin tulee pystyviiva, jota seuraa varsinainen ehto. Jos ehto täyttyy, tulos on ehtoa seuraava lauseke. Ehdot käydään läpi ylhäältä alaspäin ja ne voivat olla päällekkäisiä. Viimeiseksi voidaan määritellä *otherwise*-ehto, johon kaikki muut tapaukset täsmäytyvät. [Mukund 2007a]

Haskellissa sisennyksillä on kieliopillinen merkitys. Koodin sommittelu sisennyksillä määrää ohjelman rakenteen. Esimerkiksi *guards*-rakennetta käytettäessä pystyviivalla alkavat rivit tulee sisentää, jotta Haskell tietää, että ne ovat jatkoa aloitetulle määrittelylle eivätkä uutta määrittelyä. Sisennysten merkityksellisyys tekee puolipisteiden ja erilaisten sulkeiden käytöstä vapaaehtoista. [Mukund 2007a]

Haskellissa käytetään listoja sitomaan yhteen ryhmä arvoja. Lista on jono saman tyyppisiä arvoja, jotka on kirjoitettu hakasulkeiden sisään ja erotettu pilkulla. Listat voivat sisältää toisia listoja eli ne voivat olla sisäkkäisiä. Kaikkien listan alkioiden täytyy olla samaa tyyppiä. Esimerkiksi tyyppiä *Int* oleva lista *[1,2,3,4]* on oikeellinen, koska kaikki alkiot ovat kokonaislukuja, kun taas lista *[[1,2],3,4]* ei ole oikeellinen, koska siinä on kahta tyyppiä *[Int]* ja *Int*. [Mukund 2007a]

Haskell rakentaa listat sisäisesti alkio kerrallaan aloittaen tyhjästä listasta. Haskell toimii historiallisista syistä oikealta vasemmalle eli lisäämällä uuden alkion listan alkuun. Yksi listan rakennusoperaatio on kaksoispiste, joka ottaa alkion ja listan ja palauttaa uuden listan. Esimerkiksi *1:[2,3,4]* antaa tulokseksi *[1,2,3,4]*. Sisäisesti tuloslista on *1:2:3:4:[]* ja Haskell käsittelee sitä oikealta vasemmalle. Haskellille kaikki listat ovat sisäisesti edellisessä muodossa eli listat *[1,2,3,4]*, *1:[2,3,4]*, *1:2:[3,4]*, *1:2:3:[4]* ja *1:2:3:4:[]* ovat sama asia. [Mukund 2007a]

Listoista voidaan generoida uusia listoja Haskellin syntaktisen rakenteen avulla, jossa listan sisällä määritetään ominaisuudet, jotka sen alkioille halutaan. Tätä rakennetta kutsutaan englanninkielisellä termillä *list comprehension*. Se vastaa matematiikan joukko-opin joukon määritystapaa (set comprehension). Esimerkiksi joukko-opissa joukko, jossa on parillisten positiivisten kokonaislukujen neliöt, määritellään $\{n^2 \mid n \in \{1,2,...,m\}, 2 \mid n\}$. Vastaava lista saadaan Haskellilla seuraavasti:

```
[n^2 | n <- [1..m], mod n 2 == 0].
```
(37)

Määritys koostuu kolmesta osasta. Ensimmäisenä generoidaan lista alkuarvoista kohdassa *n <- [1..m]*. Koska Haskell käyttää laiskaa evaluointitapaa, voidaan määritellä äärettömiä listoja. Seuraavaksi suodatetaan alkuarvot pilkun jälkeisen osan mukaan eli vain parilliset luvut otetaan mukaan. Lopuksi arvot vielä käsitellään lopulliseen muotoonsa funktiolla *n*^2. [Mukund 2007a]

Haskellissa voidaan määrittää funktioita myös ilman nimiä. Näitä kutsutaan *anonyymeiksi funktioiksi* tai *lambda-abstraktioiksi*. Lambda-abstraktio tarkoittaa samaa kuin lambdafunktio. Esimerkiksi anonyymi funktio *(\x -> x^2)* on lambdafunktio $\lambda x.x^2$. Kenoviiva merkitsee lambdaa ja merkkiyhdistelmä -> vastaa lambdafunktion pistettä. Lambdakalkyylin syntaksi näkyy näinkin suoraan Haskellissa. Anonyymeja funktioita hyödynnetään esimerkiksi, kun funktiota käytetään vain kerran annettaessa se parametrina korkeamman kertaluvun funktiolle. [HaskellWiki 2016]

5. Funktionaalinen ohjelmointi opetuksessa

Matematiikan ja tietojenkäsittelyn välillä olevaa yhteyttä hyödynnetään harvoin tietojenkäsittelyn opetuksessa täysin. Siitä syystä tietojenkäsittelyn opetus on usein pelkistetty ohjelmointikielien syntaksin esittämiseen, eikä keskitytä siihen miksi tai miten ohjelmat näillä kielillä toimivat. Tämä johtunee siitä, etteivät imperatiiviset ohjelmointikielet anna sellaista riittävää mentaalista mallia tietojenkäsittelystä, joka sopii yhteen opiskelijan matematiikan tuntemuksen kanssa. Esimerkiksi imperatiivisessa ohjelmoinnissa täytyy ottaa huomioon tietokoneen laitteistoon liittyviä asioita kuten muistin käyttö, josta funktionaalisessa ohjelmoinnissa ohjelmoijan ei tarvitse erikseen huolehtia. Imperatiivisissa ohjelmissa myös lauseiden suorittaminen tietyssä järjestyksessä on tärkeää, toisin kuin matematiikassa ja funktionaalisessa ohjelmoinnissa. [Choppella *et al.* 2012]

Ohjelmoinnin johdantokursseilla tulisi opettaa yleisempiä ohjelmoinnin ja laskennan periaatteita. Kolme pääperiaatetta ovat ohjelmoinnin perustekniikoiden opettaminen, laskennan keskeisten käsitteiden esittely ja opiskelijan analyyttisen ajattelun ja ongelmanratkaisutaitojen edistäminen. Esimerkiksi algoritmeissa kiteytyy nämä kolme periaatetta. Funktionaalinen ohjelmointi ja erityisesti puhtaat funktionaaliset kielet tukevat näitä kolmea tavoitetta kevyellä syntaksillaan ja puhtaalla semantiikallaan sekä korkealla abstraktion tasollaan. Funktionaaliset kielet tukevat myös siirtymistä kieleen keskittyvästä opetuksesta enemmän käsitteisiin kohdistuvaan opetukseen. [Chakravarty and Keller 2004]

Funktionaalinen ohjelmointi antaa käytännöllisen lähestymistavan ongelman ratkaisuun yleisesti ja auttaa ymmärtämään tietojenkäsittelyn monia puolia. Se muodostaa erityisesti yhteyden laskennan formaalien menetelmien ja niiden soveltamisen välille. [Michaelson 1989]

Funktionaalinen ohjelmointi edistää laskennallista ajattelua algebrallisena manipulointina. Ohjelmat ovat kaavoja, jotka voidaan sieventää *sijoituksella* (substitution) eli yksinkertaistaa korvaamalla termejä. Kun ei voida enää sieventää, lopputulos on jäljelle jäänyt kaava kuten lambdakalkyylissä ja matematiikassa yleensäkin. [Choppella *et al.* 2012]

Funktionaalisen ohjelmoinnin opetus ohjelmoinnin johdantokursseilla on kiistanalainen aihe. Jotkut opettajat pitävät puhtaita funktionaalisia kieliä ideaalisina opetuskielinä, sillä ne eivät keskity laitteiston piirteisiin vaan yleiseen ongelman ratkaisuun. Sen takia funktionaaliset kielet saattavat olla parempia algoritmien suunnitteluun ja analysointiin perehdytyksessä. Painotus tulisi tällöin olla funktionaalisten kielten yleisissä piirteissä. [Vujošević-Janičić and Tošić 2008]

Funktionaalisessa ohjelmoinnissa on kuitenkin keskeisiä piirteitä, jotka ovat melko edistyneitä käsitteitä muissa ohjelmointiparadigmoissa, kuten korkeamman kertaluvun funktiot ja anonyymit funktiot (lambdalausekkeet). Näiden ominaisuuksien liian syvällinen opettaminen voi sekoittaa opiskelijaa. Toisaalta ne voidaan jättää pois johdantokursseista ilman, että kurssin johdonmukaisuus rikkoutuu. On kuitenkin keskeisiä, mutta edistyneempiä, funktionaalisen ohjelmoinnin piirteitä, joita ei voida jättää käsittelemättä, kuten rekursio. Imperatiivisissa kielissä hyvänä puolena on, että voidaan esitellä ensin iterointi silmukoiden ja vasta myöhemmin rekursio. [Chakravarty and Keller 2004]

Kuitenkin osa opettajista pitää lambdakalkyylin perusosaamista tärkeänä, jotta funktionaalinen ohjelmointi hallittaisiin syvällisesti. Toisaalta lambdakalkyylin omaksuminen ei ole aloittelijoille helppoa. Argumentin sijoitus täytyisi ymmärtää kunnolla, sillä se on yksi peruskäsitteistä. Myöhempi funktionaalisen ohjelmoinnin omaksuminen vaikeutuu, jos sitä ei ole ymmärtänyt oikein. [Vujošević-Janičić and Tošić 2008] Jos lambdakalkyylin ideaa edes valotetaan opiskelijoille, funktionaalinen ohjelmointi tulee heille ymmärrettävämmäksi [Vichare 2013].

Funktionaalisen kielen käyttö ohjelmoinnin opetuksessa asettaa opiskelijat samalle linjalle, sillä vaikka osalla olisikin jo ohjelmointikokemusta, funktionaaliset kielet ovat harvoille tuttuja [Chakravarty and Keller 2004]. Oman haasteensa imperatiivisilla kielillä ohjelmoinneille opiskelijoille voivat tuoda funktiot, sijoitus ja rekursio käsitteinä [Michaelson 1989]. Näin jo ohjelmointia osaavat eivät kyllästy ja vasta aloittavat eivät kadota motivaatiotaan triviaalien syntaksiongelmien kanssa painiessa, joita raskaamman syntaksin omaavat kielet voivat aiheuttaa. Myös opiskelijat, joilla on matemaattinen tausta, hyötyvät funktionaalisten kielten käytöstä ohjelmoinnin opetuksessa. [Chakravarty and Keller 2004]

Funktionaalisten kielten ilmaisuvoimaisuuden ansiosta monimutkaisia ohjelmointiharjoituksia pystytään ratkomaan aiemmassa vaiheessa verrattuna imperatiivisilla tai olio-ohjelmointikielillä opetettaessa. Tällöin voidaan keskittyä ongelmanratkaisuun eikä ohjelmoinnin tekniikoihin, mikä myös mahdollistaa mielenkiintoisempien ohjelmointiharjoitusten antamisen opiskelijoille. Funktionaalisten kielten, erityisesti Haskellin, algebrallisten tietorakenteiden määritysten ja hahmontunnistuksen ansiosta myös monimutkaiset rakenteet, kuten erilaiset puut, ovat aloittelijoidenkin käytettävissä. [Chakravarty and Keller 2004]

Funktionaalisten kielten tiivis syntaksi ja ilmaisuvoimaisuus tukevat luentoja, joissa luennoitsija opettaa kirjoittamalla interaktiivisesti melko monimutkaisia ohjelmia. Näin opiskelijat näkevät koko prosessin eivätkä vain ratkaisua. Tällainen opetustapa on todettu hyödyllisemmäksi kuin perinteisten luentokalvojen tai liitutaulun käyttö luennolla. [Chakravarty and Keller 2004]

Suurin haaste funktionaalisella kielellä opettamisella ohjelmoinnin johdantokursseilla on sen vähäinen käyttö oikeassa maailmassa, mikä aiheuttaa vastalauseita opiskelijoilta. Tämä voidaan kuitenkin ratkaista selittämällä opiskelijoille, että perustietämys on tärkeämpää kuin yksittäiset kielet, kertomalla käytetyistä funktionaalisten kielten sovelluksista, painottaen käytännön puolia korostaen yhteyksiä muihin kieliin ja kursseihin. [Chakravarty and Keller 2004]

Funktionaalisella kielellä opetettaessa ei tulisi unohtaa *I/O ohjelmointia* eli syötteiden ja tulosteiden käytön ohjelmointia, koska I/O on keskeinen aihe ohjelmoinnissa ja ilman I/O:ta funktionaalinen ohjelmointi vaikuttaa helposti epäkäytännölliseltä. I/O myös mahdollistaa vähittäisen johdatuksen imperatiiviseen ohjelmointiin. [Chakravarty and Keller 2004]

Funktionaalisen ohjelmoinnin opetus ennen imperatiivista ohjelmointia aiheuttaa epäilyjä siitä, milloin opiskelija oppii ohjelmoinnin piirteitä, joita ei käsitellä funktionaalisessa ohjelmoinnissa. Esimerkiksi ohjelmoinnin alemman tason yksityiskohtia kuten osoittimia voidaan käsitellä myöhemmin imperatiivisen ohjelmoinnin kurssilla. Myös olio-ohjelmoinnin käsitteitä pystytään opettamaan myöhemmin, sillä funktionaalisen ohjelmoinnin tekniikat sopivat yhteen olio-ohjelmoinnin kanssa. [Choppella *et al.* 2012]

Yhteenveto

Puhtaille funktionaalisille ohjelmointikielille, kuten Haskell, yleisiä piirteitä ovat funktiot, sivuvaikutuksettomuus, rekursion käyttö ja laiska evaluointitapa. Tässä tutkielmassa näitä ominaisuuksia on käsitelty sellaisenaan ja kertomalla lambdakalkyylistä, josta nämä piirteet tulevat funktionaaliseen ohjelmointiin, sekä kertomalla kielten ominaisuuksien vaikutuksista käytettyihin tietorakenteisiin.

Funktionaalinen ohjelmointi on matemaattisesti selkeää ja hyvin ilmaisuvoimaista. Funktionaaliset ohjelmat kertovat luonnollisesti mitä ne laskevat ja datan kulku ohjelmassa näkyy hyvin eikä toteutus muistuta laitteiston toimintatapaa. Myös ohjelmien kirjoittaminen sekä virheiden etsiminen ja korjaaminen helpottuvat.

Tutkielmassa on myös tarkasteltu lyhyesti funktionaalisen ohjelmoinnin hyötyjä rinnakkaisuuden ja samanaikaisuuden toteutuksessa. Funktionaalista ohjelmointia pidetään lupaavana tekniikkana näiden tekniikoiden toteutuksessa, sillä datariippuvuudet on määritetty eksplisiittisesti eikä muuttujia niiden perinteisessä merkityksessä käytetä. Tämän hetkiset kääntäjät eivät kuitenkaan pysty automaattisesti muuntamaan sarjallista ohjelmaa tehokkaaksi rinnakkaiseksi ohjelmaksi. Lähitulevaisuudelta tällaisia puoliautomaattisia kääntäjiä voidaan odottaa.

Funktionaaliset ohjelmointikielet ovat kehittyneet ajan mittaan omine piirteineen. Jokainen kieli on vienyt funktionaalista ohjelmointia eteenpäin ja tuonut siihen jotain uutta. Kielet eroavat toisistaan muun muassa evaluointitavoiltaan, tyypitykseltään ja puhtaudeltaan. Jo aiemmin kehitettyjen funktionaalisten kielten piirteitä yhdistämään tehtiin kieli Haskell, jota tässä tutkielmassa on käsitelty muita kieliä enemmän. Haskell on puhdas, laiska ja staattisesti ja monimuotoisesti tyypitetty funktionaalinen ohjelmointikieli, jonka funktiomäärityksissä hyödynnetään hahmontunnistusta.

Tässä tutkielmassa on tarkasteltu myös funktionaalisten ohjelmointikielten käyttöä ohjelmoinnin johdantokursseilla. Tällöin painotus tulisi olla funktionaalisten kielten yleisissä piirteissä, sillä ohjelmoinnin johdantokursseilla tulisi opettaa ohjelmoinnin ja laskennan yleisiä periaatteita kuten ohjelmoinnin perustekniikoita sekä edistää opiskelijan analyyttistä ajattelua ja ongelmanratkaisutaitoja. Erityisesti puhtaat funktionaaliset kielet sopivat tavoitteeseen. Kuitenkin jotkin funktionaalisten kielten keskeiset piirteet ovat edistyneempiä ja voivat olla vaikeampia oppia. Tästä muun muassa seuraa, että funktionaalisten kielten käyttö ohjelmoinnin opetuksessa on edelleen kiistanalainen aihe.

Viiteluettelo

Henk Barendregt. 1997. The impact of the lambda calculus in logic and computer science. *The Bulletin of Symbolic Logic* 3, 2, 181–215.

Manuel M. T. Chakravarty and Gabriele Keller. 2004. The risks and benefits of teaching purely functional programming in first year. *Journal of Functional Programming* 14, 1, 113–123.

Venkatesh Choppella, Hitesh Kumar, P. Manjula and K. Viswanath. 2012. From high-school algebra to computing through functional programming. In:

IEEE Fourth International Conference on Technology for Education (2012), 180–183.

HaskellWiki. 2016. Anonymous Function. https://wiki.haskell.org/Anonymous_function. Checked 13.8.2017.

HaskellWiki. 2017. Haskell in Industry. https://wiki.haskell.org/Haskell_in_industry. Checked 27.11.2017.

Konrad Hinsen. 2009. The promises of functional programming. *Computing in Science & Engineering* 11, 4, 86–90.

Paul Hudak. 1989. Conception, evolution, and application of functional programming languages. *ACM Computing Surveys (CSUR)* 21, 3, 359–411.

John Hughes. 1989. Why functional programming matters. *The Computer Journal* 32, 2, 98–107.

Konstantin Läufer & George K. Thiruvathukal. 2009. The promises of typed, pure, and lazy functional programming: Part II. *Computing in Science & Engineering* 11, 5, 68–75.

Greg Michaelson. 1989. *An Introduction to Functional Programming through Lambda Calculus.* Addison-Wesley.

Madhavan Mukund. 2007a. A taste of functional programming — 1. *Resonance: Journal of Science Education* 12, 8, 27–48.

Madhavan Mukund. 2007b. A taste of functional programming — 2. *Resonance: Journal of Science Education* 12, 9, 40–63.

Raul Rojas. 2015. A Tutorial Introduction to the Lambda Calculus. http://arxiv.org/abs/1503.09060. Checked 29.9.2016.

Abhijat Vichare. 2013. Algorithms, the λ calculus and programming. An intuitive approach. *Resonance: Journal of Science Education* 18, 4, 345–367.

Milena Vujošević-Janičić and Dušan Tošić. 2008. The role of programming paradigms in the first programming courses. *The Teaching of Mathematics* 11, 2, 63–83.

Mobiilisovellukset mielenterveysongelmien hoidossa

Noora Toimela

Tiivistelmä

Mielenterveysongelmista on tullut suuri ongelma yhteiskunnille. Ongelmaan on esitetty ratkaisua mielenterveyden hoitoon tarkoitetuista mobiilisovelluksista. Kirjallisuuskatsauksessa tuli esille lukuisia erilaisia mielenterveyssovelluksia. Sovelluksissa on hyödynnetty menestyksekkäästi puhelimen sensoreita ja puettavia lisälaitteita käyttäjän mielialan seurantaan. Sovelluksissa on myös käytetty perustana eri terapiamuotoja, päiväkirjamenetelmiä ja mielenterveyttä parantavia harjoituksia. Osa sovelluksista mahdollistaa yhteydenpidon oman terapeutin kanssa. Suurimmat haasteet sovellusten yleistymisen tiellä ovat yksityisyys ja tietoturva, jotka ovat useissa sovelluksissa puutteellisia. Sovellusten vaikutusta mielenterveysongelmiin ja oireisiin ei myöskään ole tutkittu tarpeeksi. Käytettävyys on yleisesti ottaen otettu hyvin huomioon. Kun sovellusten toimivuutta on tutkittu, on niiden todettu olevan melko tehokkaita, ainakin käytettynä terapeutin tai muun tukihenkilön tukemana.

Avainsanat ja -sanonnat: Mobiilisovellukset, mielenterveys, mielenterveyssovellukset, älypuhelimet.

1. Johdanto

Mielenterveysongelmat ovat kasvaneet huomattavasti viime vuosina. Yhteiskunnille muodostuu niistä yhä enemmän kuluja ja ne aiheuttavat kärsimystä niin ongelmia kohtaaville kuin heidän läheisilleenkin. Samalla myös mobiililaitteet, kuten älypuhelimet, ovat yleistyneet. Mielenterveysongelmien hoitoon ja hyvän mielenterveyden edistämiseen onkin ehdotettu mobiilisovellusten hyödyntämistä. Affektiivisen eli tunteita havaitsevan, tunnistavan ja tunteisiin vaikuttavan tietojenkäsittelyn avulla voidaan tarttua ongelmiin joihin perinteiset hoitomuodot eivät pysty, tukea perinteisiä hoitoja ja ennaltaehkäistä vakavia sairauksia. Vaikka puhelinten onkin sanottu aiheuttavan mielenterveysongelmia, voidaan niitä kuitenkin käyttää myös ehkäisemään ongelmia.

Tämä tutkielma on kirjallisuuskatsaus, jossa selvitän mielenterveyden hoitoon tarkoitettujen mobiilisovellusten nykytilaa. Minkälaisia sovelluksia on olemassa ja minkälaisia tutkimuksia niistä on tehty? Lisäksi selvitän sovellusten käyttöön liittyviä ongelmia. Tutkielmassa halusin tuoda esille sovellusten mahdollisuuksia tulevaisuuden sovelluskehittelyä varten.

Aloitan luvussa 2 esittelemällä yleisimpiä mielenterveyden häiriöitä ja mitä haasteita niiden perinteisessä hoidossa on. Sen jälkeen luvussa 3 kerron yleisesti mobiilisovelluksista ja mielenterveyssovelluksista. Luvussa 4 jatkan sovellusten ominaisuuksien tarkastelua aikaisempien tutkimusten avulla. Lisäksi kerron, miten puhelimen sensoreita ja lisälaitteita on hyödynnetty ja minkälaisia eri terapiamuotoihin pohjautuvia sovelluksia on olemassa. Lopuksi luvussa 5 arvioin sovelluksiin liittyviä ongelmia ja niiden toimivuutta ja tulevaisuuden näkymiä.

2. Mielenterveyden häiriöt

2.1. Yleisimmät häiriöt

Mielenterveysongelmien on todettu yleistyneen viime vuosina. Syitä on etsitty niin yhteiskunnan rakenteiden nopeasta muutoksesta kuin yleisestä asennemuutoksesta, joka on tehnyt ongelmista puhumisesta hyväksyttävämpää. Vaikka useimmat ongelmat ovatkin lieviä, ovat vakavemmatkin häiriöt lisääntyneet.

Yleisimpiä ongelmia useimmissa maissa ovat mieliala- ja ahdistuneisuushäiriöt. Esimerkiksi mielialahäiriöihin lukeutuvaa masennustilaa sairastaa jopa viidesosa väestöstä jossakin elämänsä vaiheessa ja se myös uusiutuu helposti [Toivio ja Nordling 2013]. Suomalaisen ICD-10 luokituksen mukaan masennustilan oireisiin lukeutuu muun muassa mielialan ja aktiivisuuden laskua, mielihyvän kokemisen ja itsetunnon heikentymistä sekä itsetuhoisia ajatuksia [Terveyden ja hyvinvoinnin laitos 2012]. Masennuksen hoitoon käytetään yleisesti lääkehoitoa ja terapiaa. Kognitiivinen psykoterapia on yksi yleisimmin käytetyistä hoitomuodoista ja sen tavoitteena on selkiyttää ja muuttaa potilaan masennusoireita aiheuttavia haitallisia ajatusmalleja [Toivio ja Nordling 2013].

Ahdistuneisuushäiriöihin lukeutuvat erilaiset fobiat, paniikkihäiriöt ja yleistynyt ahdistuneisuushäiriö. Yhteistä ahdistuneisuushäiriöille on niiden aiheuttamat fyysiset oireet, kuten sydämentykytys ja huimaus. Fobioissa eli pelko-oireissa henkilö voi kokea jopa voimakasta, hallitsematonta kauhua kohdatessaan pelon kohteensa. Paniikkihäiriössä ahdistuneisuusoireet eivät kohdistu tiettyyn paikkaan tai asiaan, vaan ne saattavat alkaa yllättäen paniikkikohtauksena. Fyysisten oireiden lisäksi saattaa olla epätodellisuuden tunnetta ja kuolemanpelkoa. Pelko-oireet ja masennus esiintyvät usein yhdessä. [Terveyden ja hyvinvoinnin laitos 2012.] Ahdistuneisuushäiriöihin auttaa usein terapia. Myös itsehoitokeinot, kuten rentoutumis- ja hengitysharjoitukset, liikunta ja oireiden hallinta keskittämällä ajatukset pois ahdistuksesta edistävät paranemista. Pelkotiloissa kohteen kohtaaminen ja omien uskomusten kyseenalaistaminen auttavat. [Toivio ja Nordling 2013.]

Vaikka stressiä ei luokitellakaan mielenterveyden häiriöksi, on negatiivinen stressi hyvin yleinen ja haitallinen ongelma. Pitkittyessään se voi aiheuttaa niin fyysisiä kuin psyykkisiä oireita, ja altistaa mielenterveysongelmille. Stressin hallintaan auttaa ongelmien ja tunteiden käsittely, positiivisen ajattelun omaksuminen, arjen hallinta ja rentoutuminen [Toivio ja Nordling 2013]. Yleisesti mielenterveyteen voi vaikuttaa terveellisillä elämäntavoilla, kuten liikunnan lisäämisellä ja päihteiden vähentämisellä sekä itsetuntemuksen ja vuorovaikutustaitojen lisäämisellä.

2.2. Hoidon ongelmat

Vaikka mielenterveysongelmat ovat yleisiä, on niiden hoitamisessa useita ongelmia. Ensimmäinen este on avun hakeminen. Tietoisuus mielenterveyden häiriöistä ja palveluista ei ole kaikkien saatavilla. Apua ei haeta, jos oireitaan ei tunnista häiriöksi tai oireiden ajatellaan olevan vain väliaikaisia tai lieviä. Palveluita ei myös käytetä, jos niistä ei tiedä tarpeeksi, ei uskota niiden auttavan tai hävetään avun hakemista. Myös palveluiden kalleus ja hoidon aikaavievyys nähdään ongelmina. Mojtabai ja muut [2002] osoittivat, että Yhdysvalloissa niistä jotka tunsivat apua tarvitsevansa, 59 % haki apua, ja heistä 44 % haki apua mielenterveyden ammattilaisilta. Mojtabain ja muiden näkemyksen mukaan tietoisuuden lisääminen ja positiivisempi asenne ammattiapua kohtaan auttaisivat jo paljon.

Toinen este on avun tarjoaminen. Useimmat häiriöistä kärsivät eivät saa apua, vaikka sitä haluaisivatkin. Edes maailman rikkaimmat maat eivät pysty hoitamaan kaikkia ja tilanne on luonnollisesti huonompi köyhimmissä maissa. Mielenterveysongelmien hoitoon ei panosteta samassa mittakaavassa kuin somaattisiin sairauksiin, eivätkä resurssit näin riitä. The WHO World Mental Health Survey Consortiumin [2004] tutkimuksesta käy ilmi, että vakavimmissakin tapauksissa hoitamatta on jätetty kehittyneissä maissa jopa puolet (35–50 %), ja kehittyvissä maissa vielä useampi. Keskivaikeista ja lievistä tapauksista hoitamatta on jätetty 50–88 %.

Ongelmien lisääntyessä ja hoidon tarjoamisen haasteiden takia, on ratkaisuja etsitty niin valtioiden, kuin yritystenkin taholta. Uusien menetelmien luominen on kuitenkin vaikeaa, sillä niiden tulisi olla sekä turvallisia että vaikuttavia, kustannustehokkaita, sosiaalisesti ja eettisesti hyväksyttyjä, ja niiden tulisi soveltua hyvin luonnollisiin elinympäristöihin. Menetelmiä tulisi myös testata perusteellisesti kontrolliryhmiä käyttäen. [Toivio ja Nordling 2013.] Yhdeksi ratkaisuksi hyvän mielenterveyden edistämiseen on esitetty mobiilisovellusten käyttöä hoitomuotona.

3. Mobiilisovellukset

Matkapuhelinteknologian kehittyessä myös mobiilisovellukset ovat kehittyneet ja vallanneet markkinoita. Mobiilisovellus on tietokoneohjelma, joka on suunniteltu käytettäväksi mobiililaitteella, kuten puhelimella, tabletilla tai uusimpana teknologiana älykellolla. Eri käyttöjärjestelmille on omat sovelluskauppansa, kuten Android-laitteille Google Play -kauppa, Applen laitteille Apple App Store ja Windows-laitteille Windows Phone Store. Saatavilla olevien sovellusten määrä on kasvanut räjähdysmäisesti niiden alettua tulla markkinoille vuonna 2008. Markkinoiden suurimman kaupan Play Storen sovellusten lukumäärä on jo 2,8 miljoonaa, ja toiseksi suurimalla Apple App Storella niitä on 2,2 miljoonaa. Sovellusten määrän odotetaan vain lisääntyvän, kun kehittyvä teknologia mahdollistaa niiden helpon tuottamisen. [Statista 2017.]

Keskityn tässä tutkielmassa erityisesti matka- ja älypuhelimiin. Matkapuhelimista on tullut monityökaluja viestintätarkoituksen lisäksi, ja puhelinten ominaisuudet mahdollistavat monipuolisen käytön kaikilla elämän osa-alueilla. Mobiiliteknologia eroaa merkittävästi tietokoneteknologiasta, sillä laitteisto on erilainen ja mobiililaitteilla on potentiaalia saavuttaa suurempi määrä ihmisiä. Tässä luvussa selvitän tarkemmin terveys- ja mielenterveyssovellusten tämänhetkistä tilannetta.

3.1. Mielenterveyden mobiilisovellukset

Matkapuhelinsovellusten määrän kasvaessa myös terveyssovellukset, eli mHealth-sovellukset ovat yleistyneet; vuonna 2015 terveyssovellusten määrä ylitti 165 000. Terveyssovelluksista suurin osa keskittyy hyvinvoinnin tukemiseen, ruokavalioon tai liikuntaan. Neljäsosa liittyy tiettyjen sairauksien hoitoon. Terveyssovelluksia tutkittaessa on niillä todettu olevan positiivista vaikutusta käyttäjille. [IMS Institute for Healthcare Informatics 2015.]

Kaikista terveyssovelluksista mielenterveyden hoitoon keskittyviä sovelluksia on vain noin 3 %. Määrä on kuitenkin laskettavissa tuhansissa. [IMS Institute for Healthcare Informatics 2015.] Lisäksi on paljon muun muassa tutkimuskäyttöön kehitettyjä sovelluksia, jotka eivät ole saatavilla sovelluskaupoista. Hoitoa, tukea ja tietoa tarjotaan yleisimpiin luokiteltuihin häiriöihin ja muihin ongelmiin, kuten masennukseen ja itsemurhien ehkäisyyn, stressiin, ahdistuneisuushäiriöihin, posttraumaattiseen stressihäiriöön, autismiin, skitsofreniaan, kaksisuuntaiseen mielialahäiriöön ja syömishäiriöihin.

Kiinnostus mielenterveyden hoitoon mobiilisovellusten avulla voi selittyä niiden monilla hyödyillä, joita ei tavanomaisilla hoitomuodoilla tai tietokonesovelluksilla välttämättä saavuteta. Mobiililaitteet ovat nimensä mukaisesti mobiileja, eli puhelin on usein mukana kaikkialla ja se on myös useimmiten

päällä. Apua on tarpeen mukaan saatavilla nopeasti ympäri vuorokauden. Sovellus on saatavilla myös silloin, kun oma terapeutti ei ole. Nopeasti taskusta saatava laite voi vastata mielenterveyspalveluiden haasteeseen nopeasta hoitoon pääsystä ja hoitoon hakeutumisen esteistä.

Sovellusten hyödyiksi voidaan lukea hoidon hyvä ja helppo saavutettavuus, mahdollisuus reaaliaikaiseen oireiden, aktiivisuuden ja terveydentilan seurantaan, hoidon saaminen mihin ja milloin vain, henkilökohtaisuus ja mahdollisuus hoidon etenemisen seurantaan. Sovellukset tukevat käyttäjien halua osallistua aktiivisesti oman terveyden parantamiseen. [Donker *et al.* 2013.]

Sovelluksia voidaan karkeasti jakaa ensinnäkin itsenäisesti käytettäviin ja esimerkiksi terapeutin avustuksella käytettäviin, ja toisekseen käyttäjän aktiivisuutta vaativiin ja automaattisesti tietoa kerääviin. Tällä hetkellä sovellukset keskittyvät lähinnä itsehoitoon, eikä potilaiden ja mielenterveyspalveluiden tarjoajien väliseen hoidon aikana tapahtuvaan kommunikointiin ja seurantaan löydy työkaluja. Varsinkin sovelluskaupoista löytyy lähinnä ehkäisevään mielenterveydenhoitoon liittyviä ja lievien oireiden hoitoon tarkoitettuja työkaluja.

Jako aktiivisiin ja passiivisiin sovelluksiin on usein häilyvä, sillä monissa sovelluksissa on useita ominaisuuksia ja tapoja seurata oireita ja vaikuttaa mielialaan. Käyttäjän aktiivisuutta vaaditaan esimerkiksi terapian kaltaisissa mielialapäiväkirjan täytössä, omien oireiden arvioinnissa ja sovelluksen ehdottamien harjoitusten tekemisessä. Ominaisuuksia, jotka eivät vaadi käyttäjän aktiivisuutta, on esimerkiksi puhelimen sensoreiden ja puhelimen käyttötapojen seurannan hyödyntäminen tiedon keräämisessä käyttäjän toiminnasta. GPS-paikannuksella ja puhelimen käytön perusteella saadaan tietoa käyttäjän aktiivisuuden ja mielialan muutoksista. Dataa voidaan saada myös ulkoisista laitteista, kuten sykemittarista. Näistä jaoista riippumatta sovellus voi perustua joko johonkin terapiamuotoon, esimerkiksi kognitiivis-behavioraaliseen, tai vain yleisesti oireiden lievittämiseen elämäntapojen parantamisen kautta.

Sovellusten hyväksyttävyyttä ja ihmisten halua käyttää niitä on tutkittu jonkin verran. Proudfootin ja muiden [2010] tutkimuksessa kävi ilmi, että suurin osa kyselyyn vastanneista suhtautui positiivisesti ajatukseen mielenterveytensä hoidosta mobiilisovelluksilla. Tiettyjen ehtojen täytyy kuitenkin täyttyä. Suurimpina huolenaiheina oli käyttäjien tietoturva ja yksityisyys ja siksi sovellukseen vaadittiin salasanaa tai muuta suojausta. Toisena vaatimuksena oli hyvä käytettävyys. Sovelluksen tulisi olla yksinkertainen ja helppo käyttää, eikä se saisi olla liian tungetteleva lähettämällä ilmoituksia jatkuvasti.

Myös mielenterveyspalveluiden tarjoajat suhtautuivat myönteisesti sovelluksiin. Schueller ja muut [2016] tutkivat mielenterveyden ammattilaisten ajatuksia. Monet raportoivat suositelleensa potilailleen informatiivisia internetsivuja,

mutta eivät sovelluksia, johtuen yksinkertaisesti niiden puutteesta. Selvisi, että mielenterveyden ammattilaiset olisivat erityisesti kiinnostuneita ratkaisuista, joilla potilaisiin voisi olla yhteydessä tapaamisten ulkopuolellakin. Potilaan käyttäytymisen ja oirehtimisen seurantaan ja kommunikaatioon haluttiin työkaluja. Sovellusten uskottiin myös auttavan potilaan sitoutumisessa kotitehtävien tekoon, ja psyykkisten taitojen ja strategioiden oppimiseen. Kuten käyttäjiä, myös palveluiden tarjoajia mietitytti yksityisyys, turvallisuus ja käytettävyys sekä lisäksi kulut. Teknologian toivottiin tekevän mielenterveystyöstä helpompaa ja palveluiden laadusta parempaa, eikä vaikeuttavan työskentelyä.

Seuraavassa luvussa selvitän tarkemmin saatavilla olevien sovellusten ominaisuuksia.

4. Sovellusten ominaisuuksia

Nykyaikaiset älypuhelimet ovat tehokkaita teknisiä laitteita niin suorittimeltaan, muistiltaan, akultaan kuin näytön laadultaan. Niissä on paljon erilaisia sisäänrakennettuja sensoreita kuten GPS-paikannin, kiihtyvyysanturi, liikkeentunnistussensori, mikrofoni ja valaistuksentunnistussensori. Puhelimet yhdistyvät niin 3G/4G-verkoilla, kuin Wi-Fi:llä ja Bluetoothilla. Tunteita tunnistavaa ja niihin vaikuttavaa, eli affektiivista tietojenkäsittelyä ja puhelinten ominaisuuksia voidaan käyttää apuna tunnistamaan monenlaisia mielenterveysongelmia ja auttamaan niissä.

Tässä luvussa selvitän, miten mielenterveyden mobiilisovellukset ensinnäkin keräävät tietoa käyttäjästä, ja toisekseen, miten käyttäjiä autetaan selviämään ongelmistaan. Käyn läpi, miten sovellukset itsenäisesti ja aktiivisesti keräävät tietoa puhelimen sensoreita ja lisälaitteita hyväksikäyttäen, eli miten mielialaa ja oirehdintaa voi seurata. Käyn myös läpi, minkälaisia käyttäjän aktiivisuutta vaativia, eri terapiamuotoihin perustuvia sovelluksia on olemassa, ja miten mielialaan voidaan vaikuttaa.

4.1. Sensorit ja puhelimen käytön seuranta

Teknologian kehittyessä monet sovellukset ovat alkaneet hyödyntää puhelimista löytyviä sensoreita. Myös mielenterveyden hoitoon suunnitellut sovellukset käyttävät niitä, mutta useimmat näistä on tehty vain tutkimuskäyttöön. Sensorit ovat hyvin monikäyttöisiä; niitä voidaan käyttää niin diagnostiikan ja oireiden seurannan apuna, kuin sovelluksen ja mobiiliterapioiden personointiin potilaan tilasta saatavien vihjeiden avulla.

Sensoreiden hyödyksi voidaan lukea niiden huomaamattomuus. Ne eivät välttämättä häiritse käyttäjän normaalia elämää tai puhelimen käyttöä millään tavalla, eikä käyttäjän tarvitse aktiivisesti raportoida esimerkiksi mielialansa

muutoksista. Sensoreilla on kuitenkin mahdollisuus tarjota monipuolisia ja luotettavia vihjeitä käyttäjän mielialasta ja kontekstista.

Mielialaa esimerkiksi on perinteisesti tarkkailtu psykologisten arviointien, kuten haastattelujen ja testien avulla. Ongelmana on kuitenkin mielialan subjektiivisuus. Sitä on vaikea mitata objektiivisesti ja se saattaa myös muuttua usein. Sensoreilla saatu tieto on objektiivisempaa ja siten luotettavampaa, kunhan mittaus on ollut luotettava. Esittelen seuraavaksi esimerkkejä sensoreiden käytöstä mielenterveyssovelluksissa.

Koska masennustiloja sairastavat ihmiset usein kärsivät muun muassa motivaation puutteesta, vähentyneestä aktiivisuudesta ja sosiaalisesta vetäytyvyydestä, voidaan potilaan sijaintia ja liikettä tarkkailemalla selvittää masennusoireiden esiintyvyyttä. Saeb ja muut [2015] tutkivat kehittämällään Purple Robot -sovelluksella muun muassa GPS-tietojen ja masennusoireiden yhteyttä. Tutkimus osoitti, että osallistujat joilla oli enemmän masennusoireita, suosivat tiettyjä sijainteja ja vierailivat harvemmissa paikoissa. He viettivät enemmän aikaa kotona ja liikkuivat vähemmän.

Sijaintitietoja voidaan saada GPS:n lisäksi Bluetoothin ja Wi-Fi:n välityksellä. Sijainnilla on mahdollista oireiden tunnistamisen lisäksi ennakoida käyttäjää stressaavia tilanteita. Sovellukset voisivat näissä tilanteissa esimerkiksi lähettää käyttäjälle rauhoittavia viestejä. Kaksisuuntaisessa mielialahäiriössä esiintyviä mania- ja masennusvaiheita saadaan myös tarkkailtua sijainnin avulla, sillä vaiheiden muutos näkyy selvästi liikkumisen ja aktiivisuuden muutoksina.

Myös puhelimen kiihtyvyysanturi (accelometer) on osoittautunut melko toimivaksi ja lupaavaksi. Kiihtyvyysanturi mittaa puhelimen kiertoliikkeen ja näin mahdollistaa näytön kääntymisen puhelinta käännettäessä. Sekä Osmanin [2015] kaksisuuntaisen mielialahäiriön tutkimuksessa että Garcia-Cejan ja muiden [2015] stressitutkimuksessa korrelaatio oireiden ja kiihtyvyysanturitietojen välillä oli melko korkea, 60 %:sta ylöspäin. Kiihtyvyysanturi toimii mielentilan indikaattorina, koska se antaa tietoa puhelimen käytöstä, joka kertoo taas käyttäjän mielentilasta.

Kiihtyvyysanturi antaa tietoa käyttäjän aktiivisuudesta, kuten GPS-paikanninkin. Kiihtyvyysanturi on kuitenkin varteenotettavampi vaihtoehto muutamista syistä. GPS-sijainnin hyödyntämistä käyttäjän seurannassa pidetään yksityisyydensuojalle suurempana uhkana, mutta kiihtyvyysanturi sen sijaan on melko tunkeilematon sensori. Kiihtyvyysanturi ei myöskään kuluta akkua samalla tavalla kuin GPS:n jatkuva päällä pito. [Garcia-Ceja *et al.* 2015.]

Lisätutkimusta tarvitaan, jotta käyttäytymisestä ja kiihtyvyysanturin tiedoista saadaan luotua luotettavia malleja mielialan seurantaan. Koska tietynlainen puhelimen käyttö aiheuttaa aina tietynlaista liikettä, voidaan esimerkiksi

tunnistaa, milloin käyttäjä puhuu puhelimessa tai kirjoittaa viestiä. Tällöin ei erikseen tarvitse katsoa puhelulokia. Vaikeutena mallien luomisessa oli Garcia-Cejan ja muiden [2015] tutkimuksessa kaikille sopivan mallin löytäminen. Parhaan korrelaation saivatkin käyttäjäkohtaiset mallit, mutta niiden kehittäminen on luonnollisesti hankalampaa ja aikaavievempää.

Mielenterveyden tilasta voidaan saada tietoa fysiologisen tilan kautta. Puhelimen sensoreita voidaan käyttää apuna myös tässä, jopa ilman tarvetta lisälaitteille. Sovelluksilla olisi mahdollisuus muuttaa puhelin esimerkiksi sydänsähkökäyrän mittauslaitteeksi. Iverson ja muut [2005] ovat osoittaneet, että sydämen toiminnasta voidaan nähdä, onko henkilö masentunut. Sovellus voisi tarkistaa sykkeen puhelimen kameraa hyväksikäyttäen, tai uusimmissa puhelimissa sisäänrakennetulla sykemittarilla.

Sykkeen lisäksi unen laatu antaa paljon viitteitä mielenterveydestä. Toisaalta unen pituus ja laatu vaikuttavat mielialaan, ja toisaalta taas häiriöt mielialassa vaikuttavat uneen. Unta on yleensä tarkkailtu päähän asetettavilla laitteilla, mutta niiden käyttö on usein hankalaa. Chenin ja muiden [2013] tekemä tutkimus unen tunkeilemattomasta tarkkailusta antoi monia ideoita sovelluksiin. Tutkimusta varten tehty sovellus päätteli muun muassa valosensorin avulla ympäristön valon määrästä ja puhelimen käytön aktiivisuudesta, milloin käyttäjä menee nukkumaan ja milloin hän herää, ja näin päätteli unen pituuden. Näin käyttäjän ei tarvitse joka kerta nukkumaan mennessään ja herätessään kertoa sovellukselle muuttunutta tilannetta. Lisäämällä kiihtyvyysanturin tiedot nukkujan liikkeistä yön aikana saadaan myös laadullista tietoa unesta.

Tutkimuksia on tehty myös seuraamalla puhelimen käyttöä. GPS-sijainnin ja kiihtyvyysanturin lisäksi sovellukset MoodScope [LiKamWa *et al*. 2013] ja Mood Miner [Ma *et al.* 2012] käyttivät mielialan arviointiin myös kommunikaation yleisyyden seuraamista. Aktiivisuutta arvioitiin puheluiden, tekstiviestien, sähköpostin, internetin selauksen ja sovellusten käytön tiheydellä. Korrelaatio mielialan ja kommunikaation tiheyden välillä oli Mood Minerissa 50 % ja MoodScopessa 66 %. Tarkkailemalla kommunikaation tiheyden lisäksi esimerkiksi puheluiden kestoa ja mihin aikaan päivästä soitot tehdään, voidaan saada lisätietoa muuttuneesta mielentilasta. Lisäksi internetin käytön on todettu olevan yhteydessä masennukseen ja huonompaan psykologiseen hyvinvointiin; internetin käyttö kommunikointiin viittaa alhaisempiin masennusoireisiin, kun taas internetin käyttö kaikkeen muuhun viittaa masennukseen ja sosiaaliseen ahdistukseen [Bakker *et al.* 2016].

Monia muitakin tapoja on kokeiltu, kuten puhelimen näppäilydynamiikan ja näytön tilan tutkimista. Aiemmin mainitussa Purple Robot -sovelluksessa [Saeb *et al.* 2015] hyödynnettiin näytön tilaa (päällä vai pois päältä) ja siitä saatavaa

tietoa puhelimen käytön kestosta ja toistuvuudesta. Masennusoireiden voimakkuus ja kasvanut puhelimen käyttö korreloivat voimakkaasti.

Jotta puhelimen sensoreiden käytöstä mielentilan arvioimiseen saataisiin mahdollisimman paljon hyötyä, on tietoa kerättävä isoja määriä. Apuna voidaankin käyttää erilaisia koneoppimistekniikoita ja tiedonlouhintaa oppimaan yhteyksiä mielialan ja sensoritietojen välillä. Näin voidaan tunnistaa paremmin käyttäjän tilaa. Hyödyksi on myös käyttää useita sensoreita yhden sijaan. Yksittäin käytettynä tulokset eivät välttämättä ole niin tarkkoja tai luotettavia.

Ongelmia sensoreiden käytössä on ensinnäkin niiden yksityisyyttä loukkaava luonne. Puhelimen käytön tarkkailu tai sijainnin seuranta tuntuvat monista hyvin tunkeilevilta, eivätkä he halua itseään tarkkailtavan, vaikka tiedot salattaisiinkin hyvin. Hyväksyttävyyteen pitää kiinnittää huomiota. Toiseksi haasteena on, että sensoreiden käyttö saattaa kuluttaa akkua huomattavan paljon.

Sensoreiden käytössä näkisinkin tällä hetkellä enemmän potentiaalia vakavien mielenterveysongelmien hoidossa ja kuntoutuksessa, missä potilas oikeasti hyötyy seurannasta ja tarvitsee enemmän apua. Esimerkiksi kaksisuuntainen mielialahäiriö on useissa tapauksissa vakava, eikä avohoidossa olevan potilaan tilan muutoksia mania- ja masennusjaksojen välillä voida jatkuvasti tarkkailla. Sijaintia, liikettä tai puhelimen käytön tapoja seuraamalla voi sovellus kuitenkin havaita tilan muutoksen ja ilmoittaa asiasta joko vain potilaalle itselleen, tai myös potilaan tukihenkilölle. Sensoriteknologiaa on mahdollista käyttää tulevaisuudessa myös lievempiin ongelmiin, kunhan yksityisyys ja tietoturva saadaan kuntoon.

4.2. Lisälaitteet

Kaikkea oirehdintaa ei voida mitata puhelimella, vaan avuksi on otettava lisälaitteita. Puhelin voi toimia ohjauslaitteena puettaville sensoreille tai lääketieteellisille laitteille. Mahdollisia lisälaitteita, kuten älyvaatteita ja älyasusteita, jotka yhdistyvät langattomasti älypuhelimeen, on jo paljon. Terveyssovelluksista 10 prosenttia mahdollistaa yhdistämisen lisälaitteeseen, mutta nämä ovat lähinnä fitness-sovelluksia [IMS Institute for Healthcare Informatics 2015].

Erityisesti mahdollisuus sykkeen, stressitason ja unenlaadun seurantaan toisi mielenterveyssovelluksiin tärkeää lisäarvoa, sillä fysiologisista mittauksista voidaan saada selville myös mielenterveyden tilaa. On muun muassa havaittu, että unen aikana masennuspotilaiden syke on normaalia korkeampi, mikä selittää myös potilailla havaittua aikaista heräämistä [Iverson *et al.* 2005]. Sykkeen pitempiaikaisempi ja tarkempi seuranta onnistuu helposti esimerkiksi rannekkeella, kuten Samsung Gearilla. Käyttäjän tilan selvittämisen lisäksi sovellus voisi tätä

tietoa hyödyntää ehdottaessaan esimerkiksi stressaantuneelle henkilölle rentoutumistekniikoiden käyttöä.

Esimerkki sovelluksesta, joka käyttää hyödykseen sekä puhelimen omia sensoreita että lisälaitteita, on Puiattin ja muiden kehittämä MONARCA [2011]. Se seuraa kaksisuuntaista mielialahäiriötä sairastavien fyysistä ja sosiaalista aktiivisuutta ja mielialaa. Lisälaitteina käytetään ranneketta ja älysukkaa.

4.3. Mobiiliterapiat

Mobiilisovelluksilla toteutettavat ehkäisevät ja hoitavat interventiot ovat vähemmän tungetteleva vaihtoehto sensoreille. Vaihtoehtona on myös hyödyntää sensoreista saatavaa tietoa personoimaan sovelluksen tarjoamia terapiamuotoja, ja esimerkiksi tarjoamaan apua silloin, kun käyttäjä sitä tarvitsee.

Mielenterveyden hoitoon tai ehkäisyyn keskittyvän sovelluksen olisi hyvä perustua psykologian ammattilaisten käyttämiin terapia- ja hoitomuotoihin ja hyväksi havaittuihin käytäntöihin. Näin varmistetaan, että sovellus ei ensinnäkään ole vahingollinen ja toisekseen, että siitä on mahdollisimman paljon hyötyä.

Esittelen seuraavaksi muutamia tutkimuksia eri hoitomuotoja käyttävistä sovelluksista. Osa sovelluksista on tarkoitettu tiettyihin sairauksiin ja niiden jälkihoitoon, ja osa tähtää enemmän yleisen mielenterveyden hoitoon tai ennaltaehkäisyyn. Tärkeää monissa sovelluksissa tuntui olevan etenkin itsetuntemuksen ja hyvien coping- eli selviytymiskeinojen lisääminen. Nämä ovat useissa psykoterapiamuodoissa tärkeinä pidettyjä taitoja. Sovellukset jakautuivat myös siinä, että käytetäänkö niitä ainoastaan itsenäisesti, vai yhteistyössä oman terapeutin tai tukihenkilön kanssa laajentamaan perinteistä terapiaa.

Osa mielenterveyden sovelluksista käyttää perustanaan kognitiivis-behavioraalista terapiaa. Se on yksilöity psykologinen terapiamuoto, jota pidetään yhtenä tehokkaimmista keinoista muuttaa potilaan behavioraalisia, kognitiivisia ja emotionaalisia selviytymiskeinoja. Sitä käytetään esimerkiksi masennuksen, ahdistuneisuushäiriöiden, unettomuuden ja stressin hoitoon. Kognitiivis-behavioraalis-perusteinen sovellus edistää sopeutumista normaaliin elämään, opettaa huomion siirtämistä stressiä aiheuttavista ärsykkeistä mielialaa parantaviin ärsykkeisiin, ja harjoittaa taitoa muuttaa perspektiiviä tapahtumista. [Bakker *et al.* 2016.]

Kognitiivis-behavioraaliseen terapiaan liittyy, sovelluksissakin usein esiintynyt, jonkin tyyppisen päiväkirjamenetelmän käyttö. Päiväkirjaan voidaan täyttää esimerkiksi kokemuksia, ajatuksia ja tunteita. Eräs useasti mainittu päiväkirjamenetelmä oli Csikszentmihalyin ja hänen tutkimusryhmänsä kehittämä ESM eli Experience Sampling Method. Sen ideana on pyytää päiväkirjan täyttäjää py-

sähtymään tiettyihin aikoihin päivästä ja kirjaamaan ylös sen hetkisiä tuntemuksia ja ajatuksia sekä tilannetta. Näin saadaan yhdistettyä henkilön ajankäyttöä ja tunteita, ja saadaan yksityiskohtaista tietoa päivittäisten kokemusten laadusta. Päiväkirjamenetelmä voi myös auttaa ymmärtämään henkilön mielialaa tai sosiaalisia suhteita. Muistutus päiväkirjan täytöstä voi tulla tietyn ajan välein, tietyissä tapahtumissa tai satunnaisesti. [Gaggioli *et al.* 2013.]

Päiväkirjan täytöstä on hyötyä sekä terapeutille että potilaalle. Potilaan mielialaa pystytään havainnoimaan paremmin, kun tiedot ovat tallessa päiväkirjassa. Muistot vääristyvät ja varsinkin tunteet ja tilanteet voivat olla jo unohtuneet, kun potilas seuraavan kerran tapaa terapeuttia. Kirjaamalla ylös tunteitaan, potilas voi oppia ymmärtämään paremmin tunteitaan ja ajatuksiaan, sillä itsetarkkailu lisää itsetuntemusta [Gaggioli *et al.* 2013]. Tällainen hetkittäisten tunteiden ja ajatusten ylöskirjaus voi olla parempi tapa kuvata henkilön yleistä hyvinvointia kuin esimerkiksi yleisluontoiset kysymykset onnellisuudesta.

Perinteisesti päiväkirjaa on pidetty kynällä ja paperilla, mikä saattaa olla työlästä. Varsinkin masentuneen henkilön motivaatio ottaa paperi esille ja alkaa kirjoittaa voi olla huono. Mobiilisovelluksissa täyttäminen voidaan tehdä mahdollisimman helpoksi ja nopeaksi. Automaattinen päivämäärien ja kellonaikojen lisääminen, ja esimerkiksi valmiit valintaruutu-vaihtoehdot vastauksille auttavat tekemään täyttämisestä helpompaa. Terapeutin tehtäviäkin voidaan helpottaa esimerkiksi käyttämällä algoritmia mielialan tarkkailuun, ja hälyttävien oireiden havaitsemiseen.

Morrisin ja muiden [2010] tutkimuksessa selvitettiin itsetietoisuutta lisäävän sovelluksen toimivuutta. Osallistujat raportoivat mielialaansa sovellukseen käyttämällä mielialakarttoja ja -asteikkoja.

Mobiilisovellusten hyötyjä on myös mahdollisuus välittömään päiväkirjamerkintöihin liittyvään palautteeseen. Käyttäjän vastaukset kysymykseen sen hetkisestä mielialasta vaikuttavat siihen, minkälaista tukea sovellus seuraavaksi tarjoaa. Jos käyttäjä esimerkiksi ilmaisee tuntevansa itsensä alakuloiseksi, voi sovellus ehdottaa sopivia itsehoitokeinoja tai näyttää vaikkapa kissavideoita piristämään käyttäjää. Sovelluksen tarjoama terapia voi myös keskittyä sosiaalisten taitojen harjoitteluun, jos käyttäjä on ilmoittanut kokevansa vaikeuksia sosiaalisissa suhteissa.

Päiväkirjamenetelmien lisäksi sovelluksista löytyy erilaisia harjoituksia. The Challenger App [Miloff *et al.* 2015] suunniteltiin helpottamaan sosiaalisten tilanteiden pelkoa. Käyttäjää kehotetaan suorittamaan vähitellen vaikeutuvia tehtäviä, jotka liittyvät vuorovaikutukseen muiden ihmisten kanssa oikeassa elämässä. Käyttäjä saa itse valita, millä alueilla hän tarvitsee harjoitusta. Sovellus

tarjoaa myös mindfulness-harjoituksia rentoutumiseen, mikä on tärkeää ahdistuneisuushäiriöissä. Käyttäjälle on myös tietoa sosiaalisten tilanteiden pelosta ja vertaistukea. Monet esimerkiksi posttraumaattista stressihäiriötä hoitavat sovellukset tarjoavat myös harjoituksia paniikkireaktioiden ja ahdistuksen hallitsemiseen, ja selviytymiskeinojen opetteluun.

Suurin osa kuluttajille saatavilla olevista sovelluksista on itsenäiseen käyttöön. Terapeutin kanssa yhteistyössä käytettäviä on lähinnä tutkimuskäytössä. MobileType [Reid *et al.* 2011] on esimerkki nuorille tarkoitetusta sovelluksesta, joka päiväkirjametodin tavoin tarkkailee päivittäin muun muassa mielialaa, stressiä, nukkumista ja alkoholin käyttöä. Tutkimuksessa nämä tiedot välitettiin nuorten omalääkäreille.

Jotta perinteisistä hoidoista saatu hyöty olisi mahdollisimman korkea, olisi tärkeää edistää potilaiden yleistä elämänhallintaa ja terveitä elintapoja. Elämänhallinta on tärkeää myös sairauksien ennaltaehkäisyssä, ja siksi arjenhallintaan, unenlaadun parantamiseen ja stressin vähentämiseen on saatavilla useita sovelluksia. Fyysinen terveys vaikuttaa suoraan mielenterveyteen. Esimerkiksi WellWave-sovellus [Macias *et al.* 2015] suunniteltiin edistämään skitsofreniaa, masennusta ja kaksisuuntaista mielialahäiriötä sairastavien terveyttä ja sitä kautta toipumista. Sovellus tarjoaa muun muassa tehtävälistoja sujuvoittamaan arkea, itsearviointeja, tietoa sairauksista ja motivoivia videoita. Elämänhallinnassa tärkeässä osassa on myös tavoitteiden asettaminen, mikä voi jopa auttaa muuttamaan omaa käytöstä.

Myös vertaistuki voi auttaa positiiviseen muutokseen mielenterveydessä, ja monille tällainen tuki on tärkeää. Sovelluksissa voisikin olla mahdollisuus olla anonyymisti yhteydessä muihin samoista ongelmista kärsiviin.

5. Arviointia

Tässä luvussa selvitän mielenterveyssovelluksissa esiintyviä suunnitteluhaasteita ja ongelmia, mitä aiemmissakin luvuissa on tullut vastaan. Selvitän myös sovellusten toimivuutta, eli voiko niistä olla oikeaa hyötyä sairauksien hoitoon. Lopuksi on lyhyt katsaus sovellusten tulevaisuuteen ja siihen, minkälaisia mahdollisuuksia niillä on.

5.1. Turvallisuus

Terveys- ja mielenterveyssovellukset poikkeavat monella tavalla muista sovelluksista, mikä aiheuttaa sovellusten suunnitteluun uusia haasteita. Suunnittelijoiden tulee ottaa huomioon uudenlaisia käytännöllisiä ja eettisiä näkökulmia.

Suurin esille tullut ongelma mielenterveyssovelluksissa oli tieteellisen näytön puute sovellusten toimivuudesta ja turvallisuudesta. Vain muutamia tuhansista sovelluksista on tutkittu ja useimmiten sovellusten teon lähtökohta onkin enemmän kaupallinen kuin tieteellinen, eikä sovellusten suunnittelussa ole konsultoitu mielenterveyden ammattilaisia käytännössä lainkaan [Donker *et al*. 2013]. Niistä sovelluksista joiden vaikutuksia on tutkittu, vaikuttavat useimmat olevan tarkoitettu vain tutkimuskäyttöön, eikä pitkäaikaisvaikutuksista löytynyt tietoa.

Vaikka mielenterveyden ammattilaisten keskuudessa onkin mielenkiintoa ottaa sovelluksia osaksi potilaiden hoitoa, eivät he kuitenkaan sovelluksia suosittele. Syynä on pääosin juuri tieteellisen näytön puute toimivuudesta. Useimmat tehdyt tutkimukset kehottavat lisätutkimuksiin, sillä esimerkiksi vääränlaisen terapian tai virheellisen tiedon tarjoaminen voi olla hyvinkin haitallista käyttäjälle. [IMS Institute for Healthcare Informatics 2015.] Sovellus ei saa olla haitallinen niin, että se estäisi käyttäjää hakemasta oikeaa ammattiapua.

Yhtenä ratkaisuna toimivien ja turvallisten sovellusten määrän kasvattamiseen voisi olla yhteiset standardit sovellusten suunnitteluun. Muutamia terveyssovelluskirjastoja onkin tehty, kuten Happtique ja NHS Health Apps Library. Näissäkin on kuitenkin edelleen ongelmansa, kuten seuraavaksi kerron.

On erittäin tärkeää ottaa huomioon yksityisyydensuoja ja tietoturva, kun suunnitellaan uutta sovellusta. Erityisesti terveyssovelluksissa käytettävien henkilökohtaisten potilastietojen, kuten diagnoosien, hoitojen ja päivärutiinien tiedot, tai sensoritietojen päätyminen väärin käsiin on turvallisuusriski. Henkilökohtaiset tiedot voivat päätyä vääriin käsiin ensinnäkin puhelimen varastamisen ja katoamisen myötä, ja toisekseen tietojensiirron aikana huonosti toteutettujen salausten takia.

Aiemmin mainittua NHS Health Apps Library -sovelluskirjastoa tutkiessaan Huckvale ja muut [2015] huomasivat, että sovellusten tietoturva oli melko heikolla tasolla. Kolmasosalla sovelluksista ei ollut minkäänlaisia yksityisyyskäytäntöjä, ja yksikään sovellus ei salannut puhelimeen tallennettuja henkilökohtaisia terveystietoja. Myöskään kaksi kolmasosaa sovelluksista, jotka lähettivät henkilökohtaisia tietoja internetin kautta, eivät käyttäneet tietojensalausta.

Huckvalen ja muiden [2015] tutkimuksessa tuli myös esille ongelma tietojen jakamisesta kolmansille osapuolille. Osa sovelluskirjaston sovelluksista lähetti käyttäjien tietoja muun muassa mainostajille, ilman että käyttäjältä oli tähän lupaa kysytty. Yksityisyyskäytännöt ja asetukset saattoivat myös olla hämmentäviä tai vaikeasti ymmärrettäviä.

Tärkeää sovelluksia suunniteltaessa olisikin ottaa lähtökohdaksi privacy-by-design eli sisäänrakennettu yksityisyydensuoja. Yksityisyys on otettava huomioon jokaisessa suunnitteluvaiheessa ja käyttäjästä tallennettavien tietojen määrä pitää minimoida. Käyttäjällä pitää sovellusta käyttäessään olla ymmärrys ja suostumus tietojensa käytöstä, ja heidän tulee tietää kuka tietoja käyttää ja mihin.

Tiedot tulisi salata sekä puhelimessa että palvelimelle ladattaessa niin, ettei henkilöä voida tunnistaa tiedoista. Sovelluksessa olisi hyvä myös käyttää salasanaa tai sormenjälkitunnistinta. Otettaessa mukaan sensoreista saatavaa tietoa tulevat yksityisyysasiat monimutkaisemmiksi. GPS-paikantimen tiedoilla on rikollisten esimerkiksi mahdollista seurata käyttäjän liikkeitä.

5.2. Käytettävyys

Kuten kaikissa sovelluksissa, on käytettävyys erittäin tärkeää myös mielenterveyssovelluksissa. Yleisesti ottaen huomasin, että käytettävyyteen oli kiinnitetty huomiota useimmissa tutkituissa sovelluksissa. Paljon huomiota oli kiinnitetty erityisryhmille, kuten skitsofreniaa sairastaville tai vanhemmille ihmisille tarkoitettuihin sovelluksiin.

Ikäihmisille riittää usein suurempi näyttö ja enemmän tukea sovelluksen käyttöön. Skitsofreniaa sairastavilla on usein vaikeuksia jopa tavallisilla internetsivuilla navigoimisessa, joten tavalliset sovellukset voivat myös osoittautua ongelmallisiksi. Käytettävyyden kannalta sovelluksessa tulisi olla yksinkertaisia sanoja, vähän tekstiä ja kuvia ja käytössä tulisi myös avustaa. [Ben-Zeev *et al.* 2015.]

Hyvin suunnitellun sovelluksen tulisi olla intuitiivinen, nopea ja helppo käyttää. Usein sovelluksista tehdään liian monimutkaisia ja niihin rakennetaan ominaisuuksia, joita keskivertokäyttäjä ei tarvitse. Suunnitteluun olisikin hyvä sisältyä paljon prototyyppien tekoa ja arviointia sekä käyttäjätestausta. Mielenterveyssovelluksia tehtäessä mukana olisi hyvä olla mielenterveyden ammattilaisia, jotta erityisryhmät otettaisiin huomioon käytettävyydessä. Käyttöliittymän pitää tukea sovelluksen tarjoamia hoitomuotoja, eikä huonon käytettävyyden takia vaikeuttaa niiden käyttämistä. Esimerkiksi päiväkirjamenetelmässä avuksi olisi erilaisten kosketusnäytöllä helposti käytettävien valintaruutujen ja liukunäppäimien käyttö. Voi olla vaikeaa muuttaa terapeuttinen sisältö sopivaan vuorovaikutukselliseen muotoon puhelimen ruudulle, missä niillä olisi sama parantava vaikutus kuin oikeassa terapiassa.

Tiedon tehokas visualisointi auttaa parantamaan käytettävyyttä. Tutkimusta tarvitaan tällä alueella, sillä mielenterveyssovelluksissa käyttäjistä saadun tiedon visualisointiin käytetään lähinnä perinteisiä jana-, ympyrä- tai pylväsdiagrammeja. Esimerkiksi MONARCA-sovellus [Puiatti *et al*. 2011] käyttää diagrammeja

visualisoimaan käyttäjän mielialaa, stressiä ja fyysistä ja sosiaalista aktiivisuutta. Diagrammit ja kaaviotkin toimivat, mutta tiedon visualisointiin on ehdotettu myös muita, mahdollisesti tehokkaampia vaihtoehtoja.

Gravenhorst ja muut [2015] kertovat abstraktimman tiedon visualisoinnin olevan suosittu tapa saada tieto vakuuttavampaan muotoon. Kun sovelletaan metaforia, kuten ”growing a garden of flowers” yritettäessä saavuttaa tiettyä tavoitetta, saattaa se kannustaa paremmin kuin tavallinen ja tylsä pylväsdiagrammi saavutuksista.

5.3. Muita suunnitteluhaasteita

Itse teknologiakin aiheuttaa muutamia ongelmia sovellusten tekemisessä. Koska teknologia kehittyy niin nopeaa vauhtia, ja sekä kehittäminen että toimivuuden testaaminen ovat hitaita prosesseja, voi sovellus olla jo vanhentunut sen tullessa markkinoille. Laajat satunnaistetut vertailukokeet voivat kestää jopa 3-5 vuotta, ja kun tulokset julkaistaan, voivat testattu sovellus ja päivitetty sovellus erota toisistaan huomattavasti [Kao *et al.* 2017]. Vanhentumisen lisäksi teknologia tuottaa aina muitakin haasteita, kuten laitteiden toimimattomuutta, yhteyksien katkeilua tai akun loppumista.

Useimmat saatavilla olevista sovelluksista on tarkoitettu kuluttajien itsenäiseen käyttöön. Kuten monissa muissakin omaan aktiivisuuteen nojaavissa palveluissa, käytön keskeyttäjien määrä on korkea. Harrison ja muut [2011] kuitenkin toteavat käytön jatkuvan pidempään, jos käyttö on tuettua. Mobiilisovelluksilla on mahdollisuus auttaa käyttäjää sitoutumaan itsehoitoon ja tarvittaessa etsimään ammattiapua. Koska sovellukset on tarkoitettu lähinnä itsenäiseen käyttöön, voi käyttäjien olla vaikea löytää ja valita toimivaa sovellusta. Tarvitaankin enemmän tietoisuutta sovelluksista, jotta niistä saataisiin tunnetumpia ja hyväksytympiä kaikissa ikä- ja muissa ryhmissä.

Tärkeintä suunnittelussa olisi, että psykologian ja tietojenkäsittelyn ammattilaiset tekisivät yhteistyötä, jotta sovellukseen saataisiin sekä hyvä käyttöliittymä että oikeaa terapeuttista arvoa ja eettisyyttä. Aina on tietenkin ongelmallista tehdä kaikille käyttäjille sopivaa sovellusta, sillä ihmiset ovat aina yksilöitä ja kaikki ominaisuudet eivät toimi kaikilla käyttäjillä. Kun mobiiliteknologia kehittyy ja sovelluksia on taloudellisempaa tehdä, voidaan niistä tehdä yhä turvallisempia ja toimivampia ja siten käyttää niitä sekä yksilötasolla että esimerkiksi ammattilaistasolla terapioissa.

5.4. Toimivuus

Ongelmista huolimatta sovellusten toimivuudesta ja hyödyistä on ollut näyttöä. Seuraavissa tutkimuksissa on käytetty muun muassa satunnaistettuja vertailukokeita, joissa on verrattu sovellusta käyttävää ryhmää ja kontrolliryhmää, joka ei käytä sovellusta. Näin on saatu tutkimustuloksia sovellusten vaikutuksista.

Kohdassa 4.3 esille tullut päiväkirjamenetelmä ja muu itsetarkkailu ovat osoittautuneet tehokkaiksi tavoiksi lisäämään itsetuntemusta. Nuorten mielenterveyden seurantaan kehitetty MobileType-sovellus [Reid *et al.* 2011] seurasi nuorten mielialaa ja stressitasoa kyselyillä. Satunnaistettu vertailukoe toteutettiin nuorten tavallisen mielenterveyden hoidon ohella. Tuloksia alkoi näkyä vasta kuuden viikon kuluttua tutkimuksen loputtua. Reid ja muut saivat selville, että nuorten itsetuntemus ja tietoisuus tunteista oli lisääntynyt. Nuoret olivat alkaneet kehittää itselleen parempia selviytymiskeinoja ja siten myös heidän mielenterveytensä oli parantunut.

Myös Harrisonin ja muiden [2011] kehittämä myCompass-sovellus käytti apuna itsetarkkailua. Lisäksi tutkimukseen osallistujat olivat yhteydessä sovelluksen kehittäjiin tekstiviestien välityksellä. Harrison ja muut [2011] osoittivat osallistujien stressin, ahdistuksen ja masennusoireiden lievittyneen huomattavasti. Erityisesti self-efficacy, eli uskomus omiin kykyihin hallita omaa mielenterveyttä, parani. Koska käyttö oli tuettua alusta loppuun asti, oli osallistujien sitoutuminen sovelluksen käyttöön korkea.

Morrisin ja muiden [2010] tutkimuksessa paitsi raportoitiin omista mielialoista, oli tarjolla myös terapeuttisia harjoituksia. Harjoitukset oli kohdistettu auttamaan haitallisten tilannetulkintojen muuttamisessa ja rentoutumisessa. Moni osallistuja kertoi muutoksista mielialassa. Osallistujien itsetuntemus oli lisääntynyt ja he kykenivät tunnistamaan toistuvia kuvioita käytöksessään ja mielialoissaan ja siten muuttamaan tapojaan käyttäytyä ja ajatella. Osallistujat sisäistivät nopeasti sovelluksen opit ja sovelsivat niitä omaan elämäänsä.

Positiivista palautetta sai myös WellWave-sovellus [Macias *et al.* 2015], joka kannusti käyttäjiä käymään kävelyillä. Lisäksi käyttäjille oli tukea tekstiviesteistä ja digitaalisesta kirjastosta. Käyttäjät kertoivat saaneensa sovelluksesta tukea päiviensä jäsentämiseen ja motivaatiota nousta ylös ja lähteä ulkoilemaan.

Terapiaperusteisia sovelluksia on siis tutkittu, mutta sensoreiden käytön hyödyistä mobiiliterapian apuna ei ole merkittäviä tutkimuksia. Sensoritutkimuksissa on tähän asti saatu selville henkilön mielialan ja sensoreista saadun tiedon yhteys, mutta sovellukset eivät ole tätä yhteyttä vielä paljoa hyödyntäneet. Lisätutkimusta tarvitaankin selvittämään, onko sensoreiden käytöstä merkittävää lisähyötyä terapiaan pohjautuvissa sovelluksissa. Mitä sensoritiedolla voidaan tehdä ja voiko niillä parantaa sovelluksia?

Mielenterveyssovelluksista voi siis olla todellista hyötyä, ainakin jos sovellus on hyvin tehty. Kuitenkin itsenäisesti käytettävien sovellusten vaikuttavuus ei vielä yltä terapiaan yhdistettyjen ja oman terapeutin tai muun ammattilaisen kanssa käytettyjen sovellusten vaikuttavuuteen. Tarvitaan vielä paljon tutkimusta, erityisesti vertailukokeita ja esi- ja jatkotutkimuksia. Vaikka sovellukset ovatkin tuottaneet lupaavia tuloksia, ei esimerkiksi pitkäaikaisvaikutuksia ole tutkittu käytännössä lainkaan [Donker *et al.* 2013].

5.5. Tulevaisuus

Mobiilisovelluksissa on paljon käyttämätöntä potentiaalia monien ihmisten mielenterveyden parantamisessa. Sovelluksilla pystytään säästämään rahaa ja henkistä kärsimystä. Tulevaisuudessa näen potentiaalia erityisesti sensoriteknologian laajemmalle hyödyntämiselle. Teknologian kehittyessä ja älypuhelinten ominaisuuksien lisääntyessä sovelluksissa voidaan hyödyntää uusia asioita ja tehdä niistä entistä toimivampia.

Tällä hetkellä sovellusten mahdollisuuksia ei ole kunnolla ymmärretty, mutta tulevaisuudessa niihin tullaan toivottavasti panostamaan enemmän. Algoritmien kyky havaita ja tunnistaa tunteita ja mielialoja tulee parantumaan. Älypuhelimet eivät ole vielä lähiaikoina katoamassa mihinkään, mutta esimerkiksi älykellojen lisääntyminen tuo uusia mahdollisuuksia ja haasteita tuottaa sovelluksia myös muille alustoille.

Mobiilisovelluksilla itsensä hoitaminen on tietenkin erilaista, kuin tavalliset terapia- ja lääkehoidot, ja siihen tulisikin suhtautua sen mukaisesti. Näkemykseni mukaan sovelluksia ei tulisi pitää ainoana hoitovaihtoehtona, varsinkaan vakavimmissa häiriöissä, vaan hoidon lisänä tai ensiaskeleena perinteiseen hoitoon hakeutumisessa. Hoitoresursseihin tulisi panostaa jatkossakin, eikä jättää ammattilaisten töitä puhelimen hoidettavaksi. Näiden seikkojen takia avun hakemiseen sovelluksista tulisi ainakin vielä suhtautua kriittisesti. Tulevaisuudessakin mielenterveysalan ammattilaisia tarvitaan hoidossa, puhelin tuskin tulee korvaamaan psykologeja ja terapeutteja kokonaan.

6. Yhteenveto

Tässä kirjallisuuskatsauksessa selvitin mielenterveyden hoitoon tarkoitettujen mobiilisovellusten nykytilaa. Aihe on varsin laaja ja tämä katsaus onkin vain pintaraapaisu mahdollisiin teknologioihin, ominaisuuksiin ja mahdollisuuksiin.

Mielenterveyden häiriöiden lisääntyessä on alettu etsiä keinoja mobiilisovelluksista. Sovellukset ovat hyödyntäneet puhelinten sensoreita, kuten GPS-paikanninta ja kiihtyvyyssensoria sekä tarkkailleet erilaisia puhelimenkäyttötapoja.

Lisälaitteitakin kuten ranneketta on käytetty apuna. Sovellukset ovat pohjautuneet erilaisiin terapiamuotoihin, kuten kognitiivis-behavioraaliseen terapiaan. Osa sovelluksista on tarkoitettu suoraan erilaisten häiriöiden hoitoon ja osa on tarkoitettu hyvän mielenterveyden edistämiseen elämänhallinnan kautta.

Sovellusten kehittämisessä on vielä paljon esteitä. Yksityisyys ja tietoturva eivät ole tarpeeksi hyvällä tasolla, jotta käyttäjät ja mielenterveysalan ammattilaiset niitä voisivat varauksetta käyttää. Sovellusten toimivuudesta ei myöskään ole tarpeeksi tieteellistä tutkimusta. Myös teknologia itsessään aiheuttaa ongelmia, sovellusten vanhentuessa ennen kuin niitä on ehditty testata tarpeeksi. Käytettävyys on sovelluksissa melko hyvällä tasolla ja erityisryhmiä on huomioitu suunnittelussa.

Sovellukset, joita on testattu, ovat osoittautuneet melko toimiviksi hyvän mielenterveyden edistämisessä. Varsinkin terapeutin tai muun tukihenkilön kanssa yhteistyössä käytettynä käyttäjä saa parhaan hyödyn sovelluksesta.

Tulevaisuudessa sovelluksiin tulisi panostaa lisää ja luoda yhteisiä sääntöjä sovellusten suunnitteluun. Teknologian kehittyessä älypuhelimiin saadaan uusia ominaisuuksia, joita hyödyntää sovelluksissa. Lisätutkimusta tarvitaan sekä sovellusten toimivuudesta ja pitkäaikaisvaikutuksista että sensoreiden käytöstä ja niihin liittyvistä tietoturvariskeistä.

Viiteluettelo

David Bakker, Nikolaos Kazantzis, Debra Rickwood, and Nikki Rickard. 2015. Mental health smartphone apps: Review and evidence-based recommendations for future developments. *JMIR Mental Health,* 3,1, e7.

Dror Ben-Zeev, Susan M. Kaiser, Christopher J. Brenner, Mark Begale, Jennifer Duffecy, and David C. Mohr. 2015. Development and usability testing of FOCUS: A smartphone system for self-management of schizophrenia. *Psychiatric Rehabilitation Journal*, 36, 4, 289-296.

Zhenyu Chen, Mu Lin, Fanglin Chen, Nicholas D. Lane, Giuseppe Cardone, Rui Wang, Tianxing Li, Yiqiang Chen, Tanzeem Choudhury, and Andrew T. Campbell. 2013. Unobtrusive sleep monitoring using smartphones. In: *Proc. of the 7th International Conference on Pervasive Computing Technologies for Healthcare and Workshops*, 145-152.

Tara Donker, Katherine Petrie, Judy Proudfoot, Janine Clarke, Mary-Rose Birch, and Helen Christensen. 2013. Smartphones for smarter delivery of mental health programs: A systematic review. *Journal of Medical Internet Research,* 15, 11, e247.

Andrea Gaggioli, Giovanni Pioggia, Gennaro Tartarisco, Giovanni Baldus, Daniele Corda, Pietro Cipresso, and Giuseppe Riva. 2013. A mobile data collection platform for mental health research. *Personal and Ubiquitous Computing*, 17, 2, 241-251.

Enrique Garcia-Ceja, Venet Osmani, and Oscar Mayora. 2016. Automatic stress detection in working environments from smartphones' accelerometer data: A first step. *IEEE Journal of Biomedical and Health Informatics*, 20, 4, 1053-1060.

Franz Gravenhorst, Amir Muaremi, Jakob Bardram, Agnes Grünerbl, Oscar Mayora, Gabriel Wurzer, Mads Frost, Venet Osmani, Bert Arnrich, Paul Lukowicz, and Gerhard Tröster. 2015. Mobile phones as medical devices in mental disorder treatment: An overview. *Personal and Ubiquitous Computing*, 19, 2, 335-353.

Virginia Harrison, Judith Proudfoot, Wee Pang Ping, Gordon Parker, Dusan Hadzi Pavlovic, and Vijaya Manicavasagar. 2011. Mobile mental health: Review of the emerging field and proof of concept study. *Journal of Mental Health*, 20, 6, 509-524.

Kit Huckvale, José Tomás Prieto, Myra Tilney, Pierre-Jean Benghozi, and Josip Car. 2015. Unaddressed privacy risks in accredited health and wellness apps: a cross-sectional systematic assessment. *BMC Medicine*, 13, 214.

IMS Institute for Healthcare Informatics. 2015. *Patient Adoption of mHealth*.

Grant L. Iverson, Michael B. Gaetz, Edward J. Rzempoluck, Peter McLean, Wolfgang Linden, and Ronald Remick. 2005. A new potential marker for abnormal cardiac physiology in depression. *Journal of Behavioral Medicine*, 28,6, 507-511.

Cheng-Kai Kao, and David M. Liebovitz. 2017. Consumer mobile health apps: Current state, barriers, and future directions. *Clinical Informatics in Physiatry*, 9, 5, S106-S115.

Robert LiKamWa, Yunxin Liu, Nicholas D. Lane, and Lin Zhong. 2013. MoodScope: building a mood sensor from smartphone usage patterns. In: *Proc. of the 11th Annual International Conference on Mobile Systems, Applications, and Services* (MobiSys '13), 389-402.

Yuanchao Ma, Bin Xu, Yin Bai, Guodong Sun, and Run Zhu. 2012. Daily mood assessment sased on mobile phone sensing. In: *Proc. of the Ninth International Conference on Wearable and Implantable Body Sensor Networks*, 142-147.

Cathaleene Macias, Trishan Panch, Yale M. Hicks, Jason S. Scolnick, David Lyle Weene, Dost Öngür, and Bruce M. Cohen. 2015. Using smartphone apps to promote psychiatric and physical well-being. *Psychiatric Quarterly*, 86, 4, 505-519.

Alexander Miloff, Arvid Marklund, and Per Carlbring. 2015. The challenger app for social anxiety disorder: New advances in mobile psychological treatment. *Internet Interventions*, 2, 4, 382-391

Ramin Mojtabai, Mark Olfson, and David Mechanic. 2002. Perceived need and help-seeking in adults with mood, anxiety, or substance use disorders. *Archives of General Psychiatry*. 59, 1, 77-84.

Margaret E. Morris, Qusai Kathawala, Todd K. Leen, Ethan E. Gorenstein, Farzin Guilak, Michael Labhard, and William Deleeuw. 2010. Mobile therapy: Case study evaluations of a cell phone application for emotional self-awareness. *Journal of Medical Internet Research*, 12, 2, e10.

Venet Osmani. 2015. Smartphones in mental health: Detecting depressive and manic episodes. *IEEE Pervasive Computing*, 14, 3, 10-13.

Judith Proudfoot, Gordon Parker, Dusan Hadzi Pavlovic, Vijaya Manicavasagar, Einat Adler, and Alexis Whitton. 2010. Community attitudes to the appropriation of mobile phones for monitoring and managing depression, anxiety, and stress. *Journal of Medical Internet Research*, 12, 5, e64.

Alessandro Puiatti, Steven Mudda, Silvia Giordano, and Oscar Mayora. 2011. Smartphone-centred wearable sensors network for monitoring patients with bipolar disorder. In: *Proc. of the Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, 3644-3647.

Sophie C. Reid, Sylvia D. Kauer, Stephen J. C. Hearps, Alexander H. D. Crooke, Angela S. Khor, Lena A. Sanci, and George C. Patton. 2011. A mobile phone application for the assessment and management of youth mental health problems in primary care: A randomised controlled trial. *BMC Family Practice*, 12, 1, 131.

Sohrab Saeb, Mi Zhang, Christopher J. Karr, Stephen M. Schueller, Marya E. Corden, Konrad P. Kording, and David C. Mohr. 2015. Mobile phone sensor correlates of depressive symptom severity in daily-life behavior: An exploratory study. *Journal of Medical Internet Research*, 17, 7, e175.

Stephen M. Schueller, Jason J. Washburn, and Matthew Price. 2016. Exploring mental health providers' interest in using web and mobile-based tools in their practices. *Internet Interventions*, 4, 2, 145-151.

Statista. 2017. *Number of apps available in leading app stores as of March 2017.* [online] Available at: https://www.statista.com/statistics/276623/number-of-apps-available-in-leading-app-stores/ [Accessed 2 Nov. 2017].

Terveyden ja hyvinvoinnin laitos. 2012. *Psykiatrian luokituskäsikirja*. Juvenes Print Oy.

The WHO World Mental Health survey, CONSORTIUM. 2004. Prevalence, severity, and unmet need for treatment of mental disorders in the world health organization world mental health surveys. *JAMA*, 291, 21, 2581-2590.

Timo Toivio ja Esa Nordling. 2013. *Mielenterveyden psykologia*. Edita.